

明 細 書

変換装置及び変換方法及び変換プログラム及び変換プログラムを記録したコンピュータ読み取り可能な記録媒体

5

技術分野

本発明は、プログラム変換装置及びその方法に関するものである。

10 背景技術

従来からバッチ処理を行うコボルプログラムをCORBA、COMなどの技術を用いて、ほぼ全体を手によってクライアント／サーバ型のコボルプログラムに変換しようとする試みは存在した（例えば、非特許文献1（NEC Solutions、Open
15 COBOL Factory 21／Object Partner Pro、[平成14年7月12日検索]、インターネット＜URL：<http://www.sw.nec.co.jp/cced/ocf21/objptnpro/seihin.html>＞）参照）。

図78は一連のコボルプログラムをCORBA、COMなどの技術を用いて、クライアント／サーバ型コボルプログラムに変換する動作を示した従来図である。なお、ここで変換された結果は、クライアント／サーバ型ではあるが従来のコボルプログラムでありオブジェクト指向コボルプログラムではない。

このような試みでは、従来のコボル言語の手法を用いてコーディングされたコボルプログラム100をクライアント／サーバ型のプログラムに変換する。まず、人間が一連のコボルプログラム100

25

のソースコードに記載された内容を理解する。そして、人間によってコボルプログラム 100 を内容的にまとまりのある個々のプログラムに分解する必要がある。また、CORBA や COM などと先程分解された個々のプログラムとを連携するためのインターフェースをインターフェース定義言語 (IDL: INTERFACE DEFINITION LANGUAGE) によって記述する必要があるが、その作業にも人手が必要である。非特許文献 1 では、この IDL 作成の一部のみを自動化している。

このように、コボルプログラム 100 のソースコードから最終プログラム 300 のソースコードを生成するために行なわれる変換処理は、従来においては自動化されていないか、自動化されている部分があっても手動で行なわなければならない部分があり、人間の労力を軽減する変換方法が望まれていた。すなわち、従来、メインフレーム機器 4000 などに使われていたコボルプログラム 100 の資源をクライアント/サーバシステム (分散システム) においても活用可能にするために、人的労力をかけない自動化された変換装置及び方法が望まれていた。

さらに、情報システムへの要請が集中型処理から分散型処理に変化している近年の状況を踏まえると、過去にコボルプログラムで構築した業務ロジックを再利用する方法がより一層求められていた。

本発明は、既存のプログラムを新しい技術のソフトウェアに適した構造に変換することを目的とする。

発明の開示

この発明に係る変換装置は、バッチ処理をするプログラムをソースコードの形式で記憶する記憶部と、

上記記憶部が記憶したプログラムのソースコードを1以上のまとまりある処理に区切り、区切った処理を節として節毎の役割を各節の意味情報として判断する節判断部と、

5 上記節判断部が判断した各節の意味情報に基づいて上記記憶部が記憶したプログラムのソースコードからソースコード変換のための変換情報を抽出し、抽出した変換情報に基づいてプログラムのソースコードをクライアント機器用の変換結果プログラムのソースコードとサーバ機器用の変換結果プログラムのソースコードとの2つからなる変換結果プログラムのソースコードに変換する抽出／変換部
10 とを備える。

上記抽出／変換部は、上記2つの変換結果プログラムのソースコードを、オブジェクト指向プログラムのソースコードに変換することを特徴とする。

15

上記抽出／変換部は、所定のデータ構造と手続きをもつ複数のクラスに対応した複数のオブジェクト指向プログラムのテンプレートを生成し、上記2つの変換結果プログラムのソースコードから所定のデータ構造と手続きからなる情報を複数抽出し、抽出した各情報
20 をテンプレートの対応する部分に適用することによって上記2つの変換結果プログラムのソースコードを複数のオブジェクト指向プログラムのソースコードに変換する。

この発明に係る変換装置は、バッチ処理をするプログラムをソース
25 スコードの形式で記憶する記憶部と、

上記記憶部が記憶したプログラムのソースコードを1以上のまと

まりある処理に区切り、区切った処理を節として節毎の役割を各節の意味情報として判断する節判断部と、

5 所定のデータ構造と手続きをもつ複数のクラスに対応した複数のオブジェクト指向プログラムのテンプレートを生成し、上記節判断部が判断した各節の意味情報に基づいて上記記憶部に記憶されたバッチ処理をするプログラムのソースコードから所定のデータ構造と手続きからなる情報を複数抽出し、抽出した各情報をテンプレートの対応する部分に適用することによって上記記憶部が記憶したプログラムのソースコードを複数のオブジェクト指向プログラムのソースコードに変換する。

上記変換装置は、さらに、

上記記憶部が記憶したプログラムのソースコードの役割をプログラムの意味情報として判断するプログラム判断部を備え、

15 上記抽出／変換部は、上記プログラム判断部が判断したプログラムの意味情報と上記節判断部が判断した各節の意味情報とに基づいてプログラムのソースコードからプログラムのソースコードを変換するための変換情報を抽出する。

20 上記変換装置は、さらに、

上記記憶部が記憶したプログラムを構文解析する構文解析部とを備え、

上記節判断部は、上記構文解析部によって構文解析されたプログラムに含まれる各節の意味情報を判断する。

25

上記変換装置は、バッチ処理をするコボルプログラムのソースコ

ードを変換する。

この発明に係る変換方法は、バッチ処理をするプログラムをソースコードの形式で記憶し、

- 5 上記記憶したプログラムのソースコードを1以上のまとまりある処理に区切り、区切った処理を節として節毎の役割を各節の意味情報として判断し、

- 上記判断した各節の意味情報に基づいて上記記憶したプログラムのソースコードからソースコード変換のための変換情報を抽出し、
10 抽出した変換情報に基づいてプログラムのソースコードをクライアント機器用の変換結果プログラムのソースコードとサーバ機器用の変換結果プログラムのソースコードとの2つからなる変換結果プログラムのソースコードに変換する。

- 15 この発明に係る変換プログラムは、バッチ処理をするプログラムをソースコードの形式で記憶する処理と、

 上記記憶したプログラムのソースコードを1以上のまとまりある処理に区切り、区切った処理を節として節毎の役割を各節の意味情報として判断する処理と、

- 20 上記判断した各節の意味情報に基づいて上記記憶したプログラムのソースコードからソースコード変換のための変換情報を抽出し、抽出した変換情報に基づいてプログラムのソースコードをクライアント機器用の変換結果プログラムのソースコードとサーバ機器用の変換結果プログラムのソースコードとの2つからなる変換結果プログラム
25 のソースコードに変換する処理とをコンピュータに実行させる。

この発明に係るコンピュータに実行させるための変換プログラムを記録したコンピュータ読み取り可能な記録媒体は、バッチ処理をするプログラムをソースコードの形式で記憶する処理と、

- 5 上記記憶したプログラムのソースコードを1以上のまとまりある処理に区切り、区切った処理を節として節毎の役割を各節の意味情報として判断する処理と、

- 10 上記判断した各節の意味情報に基づいて上記記憶したプログラムのソースコードからソースコード変換のための変換情報を抽出し、抽出した変換情報に基づいてプログラムのソースコードをクライアント機器用の変換結果プログラムのソースコードとサーバ機器用の変換結果プログラムのソースコードとの2つからなる変換結果プログラムのソースコードに変換する処理とを備える。

- 15 この発明に係る変換装置は、オフライン一括処理を行う手続き型のプログラムソースコードを入力データとして、このプログラムソースコードをオンライン一件別処理のプログラムに変換する第1の変換部と、このオンライン一件別処理のプログラムをクライアント／サーバ環境で動作するWebプログラムに変換する第2の変換部
20 とを備えたことを特徴とする。

- 25 上記第1の変換部は、オフライン一括処理を行うプログラムソースコードとデータの命名規則と手続きの組み方の規則を入力データとして、上記プログラムソースコードをオンライン一件別処理を行うクライアント側クラスとサーバ側クラスの2種類のクラス・プログラムに変換し、

上記第 2 の変換部は、この 2 種類のクラス・プログラムを入力し、オブジェクト指向プログラムのソースコードを生成することを特徴とする。

- 5 上記第 2 の変換部は、クライアント側クラスをモデルクラスとビュークラスとコントローラクラスとの 3 種類のクラス・プログラムに変換し、またサーバ側クラスをセッションクラスとエンティティクラスとの 2 種類のクラス・プログラムに変換することを特徴とする。

10

- 上記第 1 の変換部は、上記プログラムのソースコード内にあるデータの定義を参照し、マスタファイルの定義とトランザクションファイルの定義とを判定して、プログラムの役割とそのプログラム内の要素が一連の処理の中で果たす役割とを検出し、プログラムの役割とその要素が一連の処理の中で果たす役割を表すラベルを付与する意味付与の前処理を備えたことを特徴とする。
- 15

 上記第 1 の変換部は変換 1 プログラムを実行し、上記第 2 の変換部は変換 2 プログラムを実行し、

- 20 オフライン一括処理を行うプログラムは種類分けされ、

 上記変換 1 プログラムと上記変換 2 プログラムは、オフライン一括処理を行うプログラムの各種類ごとに対応して作成され、

- 上記第 1 の変換部と上記第 2 の変換部とは、入力したオフライン一括処理を行うプログラムの種類に対応して作成された変換 1 プログラムと変換 2 プログラムをそれぞれ実行することを特徴とする。
- 25

この発明に係る変換方法は、

オフライン一括処理を行う手続き型のプログラムソースコードを入力し、

このプログラムソースコードをオンライン一件別処理のプログラムに変換し、

このオンライン一件別処理のプログラムをクライアント／サーバ環境で動作するWebプログラムに変換することを特徴とする。

図面の簡単な説明

10 図1は、プログラム変換方法の一例を示す図である。

図2は、オフライン一括処理をオンライン一件別処理に変換する図である。

図3は、オフライン一括処理をオンライン一件別処理に変換する図である。

15 図4は、変換装置A1000の内部構成図である。

図5は、コボルプログラム100から中間プログラム200を取得する動作を示した図である。

図6は、コボルプログラム100の一例を示す図である。

図7は、コボルプログラム100を分割する流れ図である。

20 図8は、各プログラムの役割を判断する流れ図である。

図9は、入力チェックプログラム102の節の構造を示す図である。

図10は、各節の役割を判断する流れ図である。

25 図11は、マッチングプログラム107の節の構造を示す図である。

図12は、各節の役割を判断する流れ図である。

図 1 3 は、入力チェックプログラム 1 0 2 と結果出力プログラム 1 0 9 とインターフェースクラス 2 1 0 の対応図である。

図 1 4 は、マッチングプログラム 1 0 7 とソートプログラム 1 0 4 とファイルクラス 2 2 0 の対応図である。

5 図 1 5 は、プログラムの抽出、変換の詳細を示す図である。

図 1 6 は、プログラムの抽出、変換の詳細を示す図である。

図 1 7 は、入力チェックプログラム 1 0 2 の一部を示す図である

。

図 1 8 は、インターフェースクラス 2 1 0 の一部を示す図である

10

。

図 1 9 は、入力チェックプログラム 1 0 2 の一部を示す図である

。

図 2 0 は、インターフェースクラス 2 1 0 の一部を示す図である

。

15 図 2 1 は、入力チェックプログラム 1 0 2 を示す図である。

図 2 2 は、インターフェースクラス 2 1 0 を示す図である。

図 2 3 は、インターフェースクラス 2 1 0 を示す図である。

図 2 4 は、結果出力プログラム 1 0 9 を示す図である。

図 2 5 は、結果出力プログラム 1 0 9 を示す図である。

20 図 2 6 は、プログラムの抽出、変換の詳細を示す図である。

図 2 7 は、プログラムの抽出、変換の詳細を示す図である。

図 2 8 は、プログラムの抽出、変換の詳細を示す図である。

図 2 9 は、マッチングプログラム 1 0 7 を示す図である。

図 3 0 は、マッチングプログラム 1 0 7 を示す図である。

25 図 3 1 は、マッチングプログラム 1 0 7 を示す図である。

図 3 2 は、ソートプログラム 1 0 4 を示す図である。

図 3 3 は、ファイルクラス 2 2 0 のプログラムを示す図である。

図 3 4 は、ファイルクラス 2 2 0 のプログラムを示す図である。

図 3 5 は、変換装置 B 2 0 0 0 の内部構成図である。

5 図 3 6 は、オンライン一件別処理をよりオブジェクト指向的なオンライン一件別処理に変換する図である。

図 3 7 は、インターフェースクラス 2 1 0 と 3 つのクラスの対応図である。

図 3 8 は、プログラムの抽出、変換の詳細を示す図である。

図 3 9 は、プログラムの抽出、変換の詳細を示す図である。

10 図 4 0 は、プログラムの抽出、変換の詳細を示す図である。

図 4 1 は、ビュークラス 3 1 0 のプログラムを示す図である。

図 4 2 は、ビュークラス 3 1 0 のプログラムを示す図である。

図 4 3 は、制御クラス 3 2 0 のプログラムを示す図である。

図 4 4 は、モデルクラス 3 3 0 のプログラムを示す図である。

15 図 4 5 は、モデルクラス 3 3 0 のプログラムを示す図である。

図 4 6 は、ファイルクラス 2 2 0 と 2 つのクラスの対応図である

。

図 4 7 は、プログラムの抽出、変換の詳細を示す図である。

図 4 8 は、プログラムの抽出、変換の詳細を示す図である。

20 図 4 9 は、セッションクラス 3 4 0 のプログラムを示す図である

。

図 5 0 は、セッションクラス 3 4 0 のプログラムを示す図である

。

25 図 5 1 は、エンティティクラス 3 5 0 のプログラムを示す図である。

図 5 2 は、エンティティクラス 3 5 0 のプログラムを示す図であ

る。

図 5 3 は、エンティティクラス 3 5 0 のプログラムを示す図である。

図 5 4 は、プログラム変換方法の他の一例を示す図である。

5 図 5 5 は、変換装置 C 3 0 0 0 の内部構成図である。

図 5 6 は、変換装置 A、変換装置 B、変換装置 C のコンピュータ基本構成図である。

図 5 7 は、実施例の変換方法と従来の変換方法とを示す図である。

10 図 5 8 は、実施例の、オフライン一括処理からオンライン一件別処理への処理方式の変換を示す図である。

図 5 9 は、実施例の、サンプルを用いて検討した 2 段階の変換とプログラムの対応づけを示す図である。

15 図 6 0 は、実施例の、クライアント側クラス 2 1 0 への対応づけを示す図である。

図 6 1 は、実施例の、サーバ側クラス 2 2 0 への対応づけを示す図である。

図 6 2 は、実施例の、View, Controller, Model クラス 3 1 0、3 2 0、3 3 0 への分解を示す図である。

20 図 6 3 は、実施例の、Session, Entity クラス 3 4 0、3 5 0 への分解を示す図である。

図 6 4 は、実施例の、現行の集計・出力プログラム 9 0 1 の集計処理を示す図である。

25 図 6 5 は、実施例の、一件別メソッドとして変換された変換後の集計・出力プログラム 9 0 1 a の集計処理を示す図である。

図 6 6 は、実施例の、現行の集計・出力プログラム 9 0 1 の集計

処理の手続きの流れを示す図である。

図 6 7 は、実施例の、一件別メソッドとして変換された変換後の集計・出力プログラム 9 0 1 a の集計処理の手続きの流れを示す図である。

5 図 6 8 は、実施例の、集計・出力プログラム 9 0 1 のソースコードからのブレイク判定の除去による集計・出力プログラム 9 0 1 a のソースコードへの変換を示す図である。

図 6 9 は、実施例の、レコード形式変換プログラム 9 0 2 の変換の方針を示す図である。

10 図 7 0 は、実施例の、現行のレコード形式変換プログラム 9 0 2 から変換後のレコード形式変換プログラム 9 0 2 a への手続きの変換を示す図である。

図 7 1 は、実施例の、レコード形式変換プログラム 9 0 2 のソースコードからの繰返し指定の除去によるレコード形式変換プログラム 9 0 2 a のソースコードへの変換を示す図である。

15 図 7 2 は、実施例の、現行の出力指示データ作成プログラム 9 0 3 の処理を示す図である。

図 7 3 は、実施例の、変換後の出力指示データ作成プログラム 9 0 3 a の処理を示す図である。

20 図 7 4 は、実施例の、現行の出力指示データ作成プログラム 9 0 3 から変換後の出力指示データ作成プログラム 9 0 3 a への手続きの変換を示す図である。

図 7 5 は、実施例の、図 6 7 の一件別メソッド（変換後の集計・出力プログラム 9 0 1 a）の集計処理の C / S クラス 2 1 0、2 2 0 への分解を示す図である。

25 図 7 6 は、実施例の、図 7 5 の C / S クラス 2 1 0、2 2 0 の集

計処理の5つのクラス310、320、330、340、350への分解を示す図である。

図77は、実施例の、一括処理から一件別処理への変更で変化する行数の割合を示す図である。

5 図78は、従来図である。

発明を実施するための最良の形態

実施の形態1.

10 本実施の形態では、既存の資産であるコボルプログラム100のソースコードをまず中間プログラム200のソースコードに変換し、さらに、最終プログラム300のソースコードに変換する装置及び方法について説明する。

15 図1は、本実施の形態によるプログラム変換方法の一例を示す図である。コボルプログラム100はメインフレーム機器における集中型処理及びオフライン処理及び一括処理（バッチ処理）の環境で動作するコボル言語で記述されたプログラムである。このプログラムのソースコードを変換装置A1000によって中間プログラム200のソースコードに変換する。

20 中間プログラム200はクライアント／サーバシステムである分散処理及びオンライン処理及び一件別処理の環境において動作するオブジェクト指向プログラムである。中間プログラム200はインターフェースクラス210とファイルクラス220とから構成される。インターフェースクラス210はクライアント機器が処理するプログラムでありクライアント機器用の変換結果プログラムの一例である。ファイルクラス220はサーバ機器が処理するプログラム
25 であり、サーバ機器用の変換結果プログラムの一例である。このよ

うに、コボルプログラム100をインターフェースクラス210と
ファイルクラス220に分解することによって、メインフレーム機
器上で動作していたコボルプログラム100をネットワーク上につ
ながれたクライアント機器とサーバ機器を連携させて処理するシス
5 テムに活用することができる。

なお、オブジェクトとは、外界の対象領域に存在するものについ
て、データ（属性）と手続き（メソッド）を一体化して表現したも
のをいう。また、クラスとは、オブジェクトの集合を抽象化した形
で定義したものをいう。

10 中間プログラム200のソースコードは、さらに、変換装置B2
000によって最終プログラム300のソースコードに変換される
。最終プログラム300は、WEBやVBやJava（登録商標）
などに連携することが可能なオブジェクト指向プログラムの一例で
ある。最終プログラム300は、中間プログラム200よりさらに
15 オブジェクト指向性の高いプログラムである。また、最終プログラ
ム300は、ビュークラス310、制御クラス320、モデルクラ
ス330、セッションクラス340、エンティティクラス350か
ら構成されるプログラムである。

このように、既存のコボルプログラム100をオブジェクト指向
20 プログラムにまでソースコード変換することにより、インターネッ
トなどのネットワーク上に構築された分散システム上で変換後のプ
ログラムを動作させることができる。よって、既存のプログラムの
さらなる有効活用が図れる。

まず、変換装置A1000が、コボルプログラム100のソース
25 コードを中間プログラム200のソースコードへ変換する方法につ
いて説明する。

図 2 は、コボルプログラム 1 0 0 によるオフライン一括処理を変換装置 A 1 0 0 0 によって中間プログラム 2 0 0 によるオンライン一件別処理に変換する一例を示す図である。

5 左側の一括処理では、メインフレーム機器が複数の入力データをチェック、ソートし、トランザクションファイルを生成した後、旧マスタファイルとマッチング、更新して、新マスタファイルを出力するとともに、必要ならばエラーリストを出力する。変換装置 A 1 0 0 0 は、このように、コボルプログラム 1 0 0 によるオフライン一括処理をクライアント／サーバシステムで行われるオンライン一件別処理に適合させるようにプログラムのソースコードを変換する。
10 。即ち、クライアント側では入力データの入力、チェックを行ない、トランザクションとしてサーバ機器に送る。サーバ機器ではクライアント機器から送られたトランザクションとマスタファイルをマッチングしてマスタファイルを更新するとともに、その結果をクライアント機器に伝える。
15

このように、変換装置 A 1 0 0 0 によってコボルプログラム 1 0 0 のソースコードを中間プログラム 2 0 0 のソースコードに変換することにより、オンラインの一件別処理が可能となる。

次に、上述したコボルプログラムでデータの追加や、更新削除などのトランザクションをオフライン一括処理を行うコボルプログラム 1 0 0 を中間プログラム 2 0 0 に変換することによってオンライン一件別処理を可能とする方法について、図 3 を用いてさらに詳細に説明する。
20

図 3 の左側はコボルプログラム 1 0 0 によるオフライン一括処理を示し、右側は中間プログラム 2 0 0 によるオンライン一件別処理を示す。
25

まず、入力・チェックプログラム102は、トランザクションファイル101のレコードを入力チェックし、チェック済みトランザクションファイル103を生成する。この処理の内容を記述した入力・チェックプログラム102のソースコードは、変換装置A1000によって、右側のインターフェースクラス210のプログラムのソースコードに変換される。

左側の一括処理で生成されたチェック済みトランザクションファイル103のレコードは右側のトランザクションレコード203に対応するものである。

次に、左側の一括処理では、チェック済みトランザクションファイル103をソートプログラム104によってソートし、チェック・ソート済みトランザクションファイル105を生成する。このソート処理は、右側の処理には不要である。なぜなら、右側の処理はバッチ処理ではなく一件別処理であるため、トランザクションレコード203は直接マッチング処理の対象レコードとなるからである。

次に、左側の一括処理では、チェック・ソート済みトランザクションファイル105と旧マスタファイル106をマッチングプログラム107によりマッチングする。その結果、新マスタファイル108を得る。この処理に対応して右側の一件別処理では、トランザクションレコード203とマスタファイル206をマッチングする（207）処理が行われる。このマッチング処理はファイルクラス220に記述され、サーバ機器によって実行される。

最後に、左側の一括処理では、結果出力プログラム109がマッチング処理から得られた結果を出力する。この処理に対応して右側の一件別処理では、サーバ機器に結果が出力される（208）。

次に、上記左側の一括処理を行うプログラムから右側の一件別処

理を行うプログラムへ、そのソースコードを変換する変換装置 A 1 0 0 0 について説明する。

図 4 は、コボルプログラム 1 0 0 のソースコードを中間プログラム 2 0 0 のソースコードに変換する変換装置 A 1 0 0 0 の内部構成及び動作を示す図である。

変換装置 A 1 0 0 0 は変換前のコボルプログラム 1 0 0 を入力する入力部 1 1 0 0 と、入力部 1 1 0 0 が入力したプログラムを複数のプログラムに分割する分割部 1 2 0 0 と、分割された各プログラムを構文解析する構文解析部 1 3 0 0 と、構文解析したプログラムから各プログラムの内容を判断するプログラム判断部 1 4 0 0 と、プログラム判断部 1 4 0 0 が判断した各プログラム中の複数の節の内容を判断する節判断部 1 5 0 0 と、節判断部 1 5 0 0 が判断した節を用いてコボルプログラム 1 0 0 から中間プログラム 2 0 0 へ変換するために必要なデータを抽出し、抽出したデータを用いて中間プログラム 2 0 0 に変換する抽出／変換部 1 6 0 0 と、変換した中間プログラム 2 0 0 を出力する出力部 1 7 0 0 と、入力部 1 1 0 0 が入力したプログラム等を記憶する記憶部 1 8 0 0 とから構成される。なお、記憶部 1 8 0 0 は必ずしも変換装置 A 1 0 0 0 の内部に存在する必要はなく、外部記憶装置を利用してもよい。

次に、図 4 に示された各内部構成を用いてコボルプログラム 1 0 0 のソースコードを中間プログラム 2 0 0 のソースコードに変換する動作を説明する。

図 5 は、左から右に進行し、左から入力されたコボルプログラム 1 0 0 を変換して右の中間プログラム 2 0 0 を取得する動作を示した図である。

前述した通り、コボルプログラム 1 0 0 は、トランザクションフ

ファイル 1 0 1 を入力し、その内容をチェックする入力チェックプログラム 1 0 2 と、各チェック済みトランザクションレコードをソートするソートプログラム 1 0 4 と、チェック・ソート済みトランザクションファイル 1 0 5 と旧マスタファイル 1 0 6 をマッチングする
5 マッチングプログラム 1 0 7 とマッチングした結果を出力する結果出力プログラム 1 0 9 とから構成されている。ただし、入力チェックプログラム 1 0 2、ソートプログラム 1 0 4、マッチングプログラム 1 0 7、結果出力プログラム 1 0 9 のうちどれかが欠けていても構わない。

10 入力部 1 1 0 0 は、このコボルプログラム 1 0 0 を入力する。

次に、分割部 1 2 0 0 は、4 つのプログラムが 1 つにまとまった入力プログラムをそれぞれ 4 つのプログラムに分割する（S 1 2 0 0）。ただし、コボルプログラム 1 0 0 が一つのまとまったプログラムである場合には、分割部 1 2 0 0 は何も処理しない。

15 次に、構文解析部 1 3 0 0 は、分割された 4 つのプログラムに記載された各構文を解析する（S 1 3 0 0）。

次に、プログラム判断部 1 4 0 0 は、構文解析部 1 3 0 0 によって行なわれた構文解析に基づいて各プログラムの役割を判断する（S 1 4 0 0）。この判断の結果、入力チェックプログラム 1 0 2 は
20 入力データの入力とチェック、ソートプログラム 1 0 4 は複数のレコードのソート、マッチングプログラム 1 0 7 はマスタファイルとのマッチング、結果出力プログラム 1 0 9 はマッチング結果の出力という役割を果たしていることが判断される。

次に、節判断部 1 5 0 0 がプログラム判断部 1 4 0 0 によって判
25 断された各プログラム中の節の役割を判断する（S 1 5 0 0）。ここで、節とはプログラムを一又は複数のまとまりある処理に区切り

、その区切った各処理をいう。

抽出／変換部 1 6 0 0 は、節判断部 1 5 0 0 によって判断された各節の役割からコボルプログラム 1 0 0 のソースコードを中間プログラム 2 0 0 のソースコードに変換するために必要なデータをコボルプログラム 1 0 0 のソースコードから抽出し、抽出したデータを中間プログラム 2 0 0 に適合するように変換する。

その結果、インターフェースクラス 2 1 0 とファイルクラス 2 2 0 とから構成される中間プログラム 2 0 0 が生成され、出力部 1 7 0 0 によって出力される。

次に、変換装置 A 1 0 0 0 の各部の動作について説明する。

まず分割部 1 2 0 0 の動作について説明する。

図 6 は、入力部 1 1 0 0 が入力したコボルプログラム 1 0 0 の一例である。ここでは、前述したように、コボルプログラム 1 0 0 は 4 つのまとまりあるプログラムから構成されている。各プログラムの最終行には「END PROGRAM.」が存在する。

図 7 は、図 6 で示したコボルプログラム 1 0 0 を複数のまとまりあるプログラムに分割する流れ図である。この流れ図は、PAD (PROBLEM ANALYSIS DIAGRAMS) によって記載されている。

ここでは、図 6 に示したプログラムに記述された見出し「IDENTIFICATION DIVISION.」から次の見出し「END PROGRAM.」までを一つのプログラムと判断し、それぞれ別個の出力ファイルに格納する。

すなわち、まず新しい出力ファイルを開き (S 1 2 0 1)、プログラムが終わるまで以下の処理を繰り返す (S 1 2 0 2)。

プログラムを一行読み (S 1 2 0 3)、「END PROGRA

M.」があるかどうかを判断する（S 1 2 0 4）。

「END PROGRAM.」が存在しない場合には、読み込んだ1行を出力ファイルに書き出す（S 1 2 0 7）。

「END PROGRAM.」が存在する場合には、まとまるある1つのプログラムの最終と判断し、出力ファイルを閉じ（S 1 2 0 5）、新しい出力ファイルを開く（S 1 2 0 6）。

この処理を繰り返すことによって、たとえば、図6に示されたコボルプログラム100は4つのプログラム（入力・チェックプログラム102、ソートプログラム104、マッチングプログラム107、結果出力プログラム109）に分割される。

なお、図7においては見出し「END PROGRAM.」が存在するかを判断してプログラムを分割したが、入力ファイル名自体を用いて各々のファイル名から各まとまりあるプログラムに分割することも可能である。

また、分割部1200は必ずしも設ける必要はなく、構文解析部1300が直接入力されたプログラムに基づいて構文解析してもよい。

次に、分割部1200によって分割された4個のプログラムの構文解析について説明する。構文解析部1300は、分割された個々のプログラムを別個に構文解析する。その結果、各プログラムについてプログラムの階層構造を表現した構文解析木が得られる。

この構文解析木の各ノードには、プログラム中の命令に対する構文上の意味情報が付与されている。この構文解析木を作る操作はプログラム判断部1400によって行なわれる。ここで、ノードとは、命令文の始まり及び終わりを境とし、プログラム中で意味を持つ最小の単位となる一連の語をまとめて格納したものをいう。

プログラム判断部 1 4 0 0 では各プログラム名の命名規則をもとに、入力チェックやマッチングなど、各プログラムの役割を判断し、判断した役割を意味情報として各プログラムに付与する。

ここで、プログラム名の命名規則について説明する。

- 5 プログラム中のプログラム名、ファイル、及びデータ項目については、予め各社内毎のコーディング規約によって命名規則が定められている。プログラム名については一連の処理に共通する名前と、入力、ソート、更新、結果出力の内どれかが識別できる名前が付いている。また、ファイル名にも各ファイルの役割を識別できる接辞が付与されている。
- 10

また、データ項目名には、ファイルのデータ項目ならばファイル名と同様にデータ項目の役割を識別できる接辞が付与されている。

これらプログラム中の命名規則に基づいて各プログラムに意味情報を付与することが可能である。

- 15 プログラム判断部 1 4 0 0 がプログラム名から各プログラムの役割を判断する流れ図を図 8 に示す。このフローは、プログラム判断部 1 4 0 0 によって実行される。

- まず、プログラム判断部 1 4 0 0 は、構文解析部 1 3 0 0 によって解析された各プログラムについて (S 1 4 0 1)、プログラム名
- 20 を抽出する (S 1 4 0 2)。

プログラム名が入力チェックならば、プログラムに意味情報として「入力チェック」を付与する (S 1 4 0 4)。

プログラム名がソートならばプログラムに意味情報「ソート」を付与する (S 1 4 0 5)。

- 25 プログラム名がマッチング更新であるならば、プログラムに意味情報「マッチング更新」を付与する (S 1 4 0 6)。

プログラム名が結果出力ならばプログラムに意味情報「結果出力」を付与する（S 1 4 0 7）。

次に、節判断部 1 5 0 0 が各プログラムの構文解析木の各節のノードを 1 つずつ抽出し、そのノードへ意味情報を付与する処理について説明する。

各プログラムは、前述したように、プログラム中のまとまりある処理である節から構成されている。

図 9 は、入力チェックプログラム 1 0 2 の各節の構造を示している。四角で囲まれた各処理が節に相当する。図 9 では、左の節が右の節を呼び出す構造になっている。

主処理（S 1 2 9）はトランザクション入力処理（S 1 3 8）と変換コントロール処理（S 1 3 5）を呼び出している。トランザクション入力処理は、トランザクションファイル 1 0 1 からレコードを読み込む処理であり、変換コントロール処理は変換コントロール処理が呼び出す各処理（変換処理、トランザクション入力処理）を繰り返す制御処理である。

変換コントロール処理（S 1 3 5）で行なわれる繰り返し処理は、変換処理（S 1 3 9）とトランザクション入力処理（S 1 3 8）を呼び出すことにより行なわれる。変換処理が行なわれた後には、チェック済トランザクション出力処理（S 1 4 1）の節によってチェック済トランザクションレコードをファイルに書き出す処理を行う。なお、S 1 3 9 の変換処理では、トランザクションレコードをチェックし、正しいレコードを出力側に転記する処理を行なっている。

このようにして、節判断部 1 5 0 0 は、図 9 で示す入力・チェックプログラム 1 0 2 の節の構造から各節の役割を判断する。

節判断部 1 5 0 0 が図 9 に示した各節の役割を判断する流れ図を図 1 0 に示す。

まず、節判断部 1 5 0 0 は節のノードを一つずつ抽出し（S 1 5 0 1）、OPEN 文又は READ 文又は WRITE 文を含むかを判断する（S 1 5 0 2）。

節判断部 1 5 0 0 は、節に対応する構文解析木のノードに OPEN 文を含むと判断されている場合には節の役割として意味情報「主処理」を付与する（S 1 5 0 3）。READ 文を含む場合には節の役割として意味情報「トランザクション入力」を付与する（S 1 5 0 4）。WRITE 文を含む場合には節の役割の意味情報として「チェック済トランザクション出力」を付与する（S 1 5 0 5）。

次に、節判断部 1 5 0 0 は、他の節に対してチェック済出力を PERFORM 文で読み出しているかを判断し（S 1 5 0 7）、呼び出している場合にはその節の役割として意味情報「変換処理」を付与する（S 1 5 0 8）。

節判断部 1 5 0 0 は、他の節に対して変換処理を PERFORM 文で呼び出している判断される場合には（S 1 5 1 0）、その節の役割として意味情報「変換処理コントロール」を付与する（S 1 5 1 1）。このようにして、各節の役割として意味情報を自動的に付与することができる。

次に、節判断部 1 5 0 0 がマッチングプログラム 1 0 7 に対して節の判断と意味情報の付与を行う動作を説明する。

図 1 1 は、マッチングプログラム 1 0 7 の節の構造を示している。図 9 と同様に四角で囲まれた処理は節を表し、左の処理が右の処理を呼び出す関係となっている。主処理（S 1 5 7）はトランザクションを入力し（S 1 6 5）、旧マスタファイルを入力し（S 1 6

6)、これらのデータを元に更新コントロールに基づいて更新処理を繰り返す。すなわち、更新コントロールでは、トランザクションキーとマスタキーを照合する合致処理を行ない(S 1 5 9)、また次のマスタレコードを準備するマスタ処理(S 1 6 9)と次のトランザクションレコードを準備するトランザクション処理(S 1 6 0)に基づいて合致処理(S 1 5 9)を繰り返す。その結果は新マスタファイルに出力される。このようにマッチングプログラム107は図11に示す節の構造を持っているので、節判断部1500はこれらの節の構造から各節の役割を判断する。

10 節判断部1500が図10に示した各節の役割を判断する流れ図を図12に示す。

まず、節判断部1500は、節が対応する構文解析木のノードを一つずつ抽出し(S 1 5 2 1)、OPEN文、トランザクションのREAD文、旧マスタファイルのREAD文、新マスタファイルのREAD文があるかどうかを判断する(S 1 5 2 2)。OPEN文がある場合には、その節の役割として、意味情報「主処理」を付与する(S 1 5 2 3)。トランザクションのREAD文がある場合にはその節に意味情報「トランザクション入力」を付与する(S 1 5 2 4)。旧マスタファイルのREAD文がある場合には、その節に意味情報「旧マスタファイル入力」を付与する(S 1 5 2 5)。新マスタファイルのREAD文がある場合にはその節に意味情報「新マスタファイル出力」を付与する(S 1 5 2 6)。

また、節判断部1500は、残りの節について、前述したプログラム中の命名規則に従ってプログラム名、ファイル名、データ項目名を用いてどのレコード定義がトランザクションファイル、新マスタファイル、旧マスタファイルのいずれかのレコードかを判断する

ことができる（S 1 5 2 8）。

以上、節判断部 1 5 0 0 が入力・チェックプログラム 1 0 2 とマッチングプログラム 1 0 7 について節の判断を行なった後、抽出／変換部 1 6 0 0 が各節の判断に基づいて中間プログラム 2 0 0 を生成するのに必要なデータを抽出し、変換する。次に、抽出／変換部 1 6 0 0 で行なわれる抽出／変換処理について説明する。

まず、前処理として中間プログラム 2 0 0 に予め各クラス中のメソッドを作り込む作業が必要である。ここで、メソッドとは具体的処理方法（手段）をいう。

10 抽出／変換部 1 6 0 0 は、この前処理として中間プログラム 2 0 0 を構成するインターフェースクラス 2 1 0 及びファイルクラス 2 2 0 の 2 つ構造を作り、各クラスにファイル名情報をつけておく。

また、上記 2 つのクラスには各々次の段落に示すメソッドを作り、中身は空にしておく。各メソッドには意味情報を付与しておき、
15 後で抽出及び変換する際に各メソッドを特定できるようにする。

図 1 3 は、入力チェックプログラム 1 0 2 と結果出力プログラム 1 0 9 とインターフェースクラス 2 1 0 の対応図である。

図 1 4 は、マッチングプログラム 1 0 7 とソートプログラム 1 0 4 とファイルクラス 2 2 0 の対応図である。

20 図 1 3 に示すように、抽出／変換部 1 6 0 0 は予め、インターフェースクラス 2 1 0 に画面表示入力メソッド（メソッド：displayScreen）とUIメインメソッド（メソッド：uiMain）と入力チェックメソッド（メソッド：changeModel）を作っておく。

25 また、抽出／変換部 1 6 0 0 は予め、ファイルクラス 2 2 0 にマッチング更新メソッド（メソッド：updateRecord）を

作っておく。

このような前処理を行なった後、図 1 3 に示すように、抽出／変換部 1 6 0 0 は、入力チェックプログラム 1 0 2 と結果出力プログラム 1 0 9 からデータを抽出、変換し、クライアント機器 5 0 0 0 用の変換結果プログラムとしてインターフェースクラス 2 1 0 を生成する抽出変換処理を行う。また、図 1 4 に示すように、抽出／変換部 1 6 0 0 はマッチングプログラム 1 0 7 とソートプログラム 1 0 4 からデータを抽出、変換し、サーバ機器 6 0 0 0 用の変換結果プログラムとしてファイルクラス 2 2 0 を生成する抽出変換処理を行う。

以下に、上記抽出／変換部 1 6 0 0 が行う抽出変換処理を説明する。

・入力チェックプログラム 1 0 2 と結果出力プログラム 1 0 9 からインターフェースクラス 2 1 0 への抽出変換処理

図 1 3 に示すように、抽出／変換部 1 6 0 0 では入力チェックプログラム 1 0 2 と結果出力プログラム 1 0 9 とからインターフェースクラス 2 1 0 を生成するために対応付けを行う。

抽出／変換部 1 6 0 0 は入力チェックプログラム 1 0 2 から入力チェックのロジックを抽出し、インターフェースクラス 2 1 0 に対応付ける。また、抽出／変換部 1 6 0 0 は結果出力プログラム 1 0 9 から伝票定義を抽出し、インターフェースクラス 2 1 0 に対応付ける。

抽出／変換部 1 6 0 0 は、次に示す 3 つのロジックの対応付けに基づいて抽出変換処理を行う。

第 1 に入力チェックプログラム 1 0 2 では、トランザクションファイル 1 0 1 からレコードを読み込むロジックが存在していたが、

インターフェースクラス 2 1 0 では画面からデータを入力してレコード形式に格納するロジックに変換する必要がある。すなわち、トランザクションファイル 1 0 1 からレコードを読み込むロジックは不要となる。したがって、第 1 の対応付けから入力チェックプログラム 1 0 2 に存在していたトランザクションファイル 1 0 1 からレコードを読み込むロジックを無視する。

第 2 に入力チェックプログラム 1 0 2 では読み込んだ各レコードのデータをチェックするロジックが存在するが、インターフェースクラス 2 1 0 においてもこのロジックを継承する必要がある。

第 3 に入力チェックプログラム 1 0 2 ではデータが正しければレコードをチェック済トランザクションファイル 1 0 3 に書き出すロジックが存在していたが、インターフェースクラス 2 1 0 においてはデータが正しければレコードをファイルクラス 2 2 0 に送るロジックに変換する必要がある。すなわち、データが正しければレコードをチェック済トランザクションファイル 1 0 3 に書き出すロジックは不要となる。したがって、第 3 の対応付けから入力チェックプログラム 1 0 2 に存在していたチェック済トランザクションファイル 1 0 3 の定義部分とチェック済トランザクションファイル 1 0 3 に書き出すロジックを無視し、レコードをチェックして正しければファイルクラス 2 2 0 のメソッドに送るようにするロジックを加える。

また、一括処理から一件別処理へ処理方法が変更されるので、入力チェックプログラム 1 0 2 に存在していた次のレコードを準備する繰り返しのロジックを無視する。

これらの対応付けに基づいたプログラムの抽出、変換の詳細を図 1 5 及び図 1 6 に示す。図 1 5 は入力チェックプログラム 1 0 2 か

らインターフェースクラス 2 1 0 のプログラムへの抽出及び変換の詳細を説明する図である。図 1 6 は結果出力プログラム 1 0 9 からインターフェースクラス 2 1 0 のプログラムへの抽出及び変換を説明する図である。図 1 5 及び図 1 6 に示す位置番号は図 1 3 及び図 1 4 に示す対応付けの番号に対応している。すなわち、入力チェックプログラム 1 0 2 の見出し部にかかれたプログラム名からインターフェースクラス 2 1 0 の見出し部のクラス名及び環境部のファイルクラス 2 2 0 へのリポジトリ指定及びクラス終わり見出しの抽出変換を行うが、その詳細については図 1 5 の位置番号（1－1）に示されている。

この抽出変換の詳細について、図 1 7 と図 1 8 を用いて説明する。

図 1 7 は、入力チェックプログラム 1 0 2 の一部を示す図である。図 1 8 は、インターフェースクラス 2 1 0 の一部を示す図である。

図 1 7 の位置番号（1－1）で示されたステップから図 1 8 の（1－1）で示されたステップが抽出変換される。

さらに、具体的に説明すると、図 1 7 の見出し部（S 1 2 1）中の PROGRAM－ID（S 1 2 2）の行から「在庫マスタ修正－入力チェック」を抽出し、図 1 8 のインターフェースクラス 2 1 0 中の見出し部（S 2 2 1）のクラス ID「在庫マスタ修正 UI」として変換する。

また、インターフェースクラス 2 1 0 内の環境部（S 2 2 2）に記載されたファイルクラス 2 2 0 に対するリポジトリ指定の内部名として、入力チェックプログラム 1 0 2 から抽出した「在庫マスタ修正」にファイルクラス名の接辞を付加するとともに、外部名につ

いては各会社毎のコーディング規約中の命名規則に従ってファイル
クラス 2 2 0 のファイル名を挿入する。

クラスの終わりを示す見出しについては後述する。

次に、図 1 7 で示す入力チェックプログラム 1 0 2 のデータ部（
5 S 1 2 3）のトランザクションファイル定義に基づき、インターフ
ェースクラス 2 1 0 のクラス変数を抽出変換する。具体的には図 1
7 に示された位置番号（1 - 2）から抽出し、図 1 8 のデータ部（
S 2 2 3）に示された位置番号（1 - 2）に変換する。このように
入力チェックプログラム 1 0 2 のトランザクションファイルの接辞
10 がついたレコードをインターフェースクラス 2 1 0 のクラス変数と
して記載し、画面からの入力をレコードの形式にしてファイルクラ
ス 2 2 0 に送ることが可能となる。

次に、入力・チェックプログラム 1 0 2 中の手続き部からインタ
ーフェースクラス 2 1 0 の入力チェックメソッドを作成するための
15 抽出変換処理について説明する。

図 1 9 は、入力チェックプログラム 1 0 2 の手続き部を示してい
る。図 2 0 は、インターフェースクラス 2 1 0 の入力チェックメソ
ッドの手続部とクラスの終わりの見出し部分を示している。

図 1. 3 に示すように、入力チェックプログラム 1 0 2 の手続き部
20 （S 1 2 8）はインターフェースクラス 2 1 0 の入力チェックメソ
ッド「changeModel」（S 2 2 7）の手続き部に変換さ
れる。各変換処理についての詳細は、図 1 5 の位置番号（1 - 4）
から（1 - 8）に示されているが、この変換処理について図 1 9 と
図 2 0 を用いて実際の具体的抽出変換を説明する。

25 図 1 9 の手続き部（S 1 2 8）中の主処理（S 1 2 9）に記載さ
れた主処理節（S 1 3 0 から S 1 3 4）を抽出し、この抽出した要

素から図 2 0 に示す入力チェックメソッドの手続き部の主処理（S 2 2 8）の処理内容を生成する変換処理を説明する。なお、抽出時の主処理部分の特定の仕方については前述したとおりであるので、省略する。

- 5 実際の構造及び構文上の変換については、まず、図 1 9 で示すファイルの O P E N 文（S 1 3 0）及びファイルの C L O S E 文（S 1 3 3）を無視する。また、変換処理コントロールへの P E R F O R M 文（S 1 3 2）から繰り返しの指定（U N T I L 指定）を無視する。ファイルの O P E N 文及び C L O S E 文を無視するのは、イ
10 ンターフェースクラス 2 1 0 では入力は画面で行ない、チェック済レコードはファイルクラス 2 2 0 に出力するために不要となるためである。また、繰り返しの指定を無視するのは、中間プログラムではレコードを 1 件しか処理しないためである（一件別処理）。

- 15 次に、S T O P R U N .（S 1 3 4）を中間プログラムの構文上必要な E X I T M E T H O D . に変換する。これは図 2 0 の S 2 3 8 に示されている。

次に、位置番号（1 - 5）に示される処理の変換について説明する。

- 20 図 1 9 において、変換処理コントロール（S 1 3 5）に記載された具体的内容（S 1 3 6、S 1 3 7）を抽出し、トランザクション入力への P E R F O R M 文（S 1 3 7）を消去する。これは、中間プログラムではレコードを 1 件しか処理しないので、次のトランザクションレコードを入力する必要がないためである。その結果、図 2 0 には S 2 3 1 及び S 2 3 2 のみが抽出される。

- 25 次に、位置番号（1 - 6）の抽出変換について説明する。

図 1 9 の S 1 3 8 で示すトランザクションファイルの R E A D 文

を無視し、処理を何も行わないことを表すCONTINUE文に置き換える。中間プログラムでは画面からの入力データが直ちにトランザクションレコードに入るため、READ文は不要となるためである。その結果、トランザクション入力処理は図20のS234で示す部分のように変換される。

次に位置番号（1－7）の変換処理についての抽出変換を説明する。

図19のS139で示す変換処理の内、S142のMOVE文、すなわち、トランザクションレコードからチェック済トランザクションレコードへレコードを移動させるための文を無視する。中間プログラムによれば、トランザクションレコードをチェックしてデータが正しければそのままファイルクラス220にそのデータを送るためである。その結果、変換処理には図20のS235で示す部分のみが抽出される。

次に、位置番号（1－8）の抽出変換処理について説明する。

ここでは、図19のS141に示すトランザクション出力処理の抽出変換処理が行なわれる。具体的にはチェック済トランザクションファイルへのWRITE文（S143）をファイルクラス220のマッチング更新メソッドへのINVOKEL文に置き換える。中間処理プログラムにおいては、チェックしたレコードはファイルクラス220のマッチング更新メソッドへの引数となるからである。このようにして、抽出変換した結果を図20のS236に示す。

なお、S237は位置番号（1－1）に対応してクラスの終わりの見出しを抽出変換した変換後のステップを示している。

以上のようにして抽出／変換部1600はコボルプログラム1000に記載された入力チェックプログラム102のソースコードから

中間プログラム 200 を構成するインターフェースクラス 210 のソースコードを自動的に抽出変換する。

5 実際の入力チェックプログラム 102 のソースコードを図 21 に示す。また、入力チェックプログラム 102 のソースコードから抽出／変換部 1600 によって抽出、変換されたインターフェースクラス 210 のソースコードを図 22、図 23 に示す。すなわち、図 22 の第 1 行であるステップ 1 (000001) から図 23 の最終行であるステップ 100 (000100) までのソースコードが抽出、変換されたインターフェースクラス 210 のソースコードである。

10 図 21、図 22、図 23 に示すプログラム中、各位置番号はどのように変換がなされたかを説明するためのものであり、実際のプログラムに記載する必要はない。また、上記において説明されていない位置番号については後述する。

15 次に、図 16 に示す結果出力プログラム 109 からインターフェースクラス 210 への抽出、変換の具体的動作を説明する。図 24、図 25 に結果出力プログラム 109 のプログラムを記載する。図 24 の第 1 行であるステップ 1 (000001) から図 25 の最終行であるステップ 84 (000084) までのソースコードが結果出力プログラム 109 のソースコードである。

20 上記結果出力プログラム 109 に記載された位置番号 (1-25) から伝票定義を抽出し、図 18 の位置番号 (1-25) に示す画面表示メソッドの画面定義に変換する。この場合、伝票の行位置、桁位置を画面の行位置、桁位置に対応付ける必要がある。

25 また、上記結果出力プログラム 109 の位置番号 (1-25) に示す伝票項目の「SOURCE 指定」を図 18 の位置番号 (1-2

5) に示す画面項目の「T O 指定」に置き換えることが必要となる。

さらに、トランザクションレコードに含まれているがマスタレコードに含まれないデータ項目をもとにそれらの項目について画面項目を生成する。結果出力プログラム 1 0 9 はマスタレコードの項目を出力しているが、インターフェースクラス 2 1 0 の画面表示メソッドでは画面からトランザクションの項目を入力するためである。

以下に、上記抽出／変換部 1 6 0 0 がマッチングプログラム 1 0 7 とソートプログラム 1 0 4 からファイルクラス 2 2 0 へ変換する情報を抽出する抽出変換処理を図 1 4 を用いて説明する。

・マッチングプログラム 1 0 7 とソートプログラム 1 0 4 からファイルクラス 2 2 0 への抽出変換処理

次に、マッチングプログラム 1 0 7 とソートプログラム 1 0 4 とに基づいて中間プログラムのファイルクラス 2 2 0 を抽出、変換する動作について説明する。前述したように、図 1 4 は左側に示すマッチングプログラム 1 0 7 とソートプログラム 1 0 4 から必要なデータを抽出し、抽出したデータに基づいてマッチングプログラム 1 0 7 またはソートプログラム 1 0 4 を変換し、右側に示すファイルクラス 2 2 0 を生成するための各対応付けを示している。

マッチングプログラム 1 0 7 とソートプログラム 1 0 4 とから中間プログラム 2 0 0 を生成するための各対応についての基本的方針を説明する。

まず、マッチングプログラム 1 0 7 からはマッチング更新のロジックを抽出する。また、ソートプログラム 1 0 4 からはレコードキーの情報を抽出する。

マッチング更新のロジックは次の 3 つの対応付けにより変換する

第1はマッチングプログラム107ではチェック・ソート済みトランザクションファイル105からレコードを読み込んでいたが、ファイルクラス220ではトランザクションレコードを手続きの引
5 数として受け取るように変換する必要がある。

第2にマッチングプログラム107ではマスタレコードとチェック・ソート済みトランザクションファイル105のレコードをマッチングするロジックが存在していたが、ファイルクラス220においてもこのロジックが必要となる。

10 第3にマッチングプログラム107ではマッチングした結果、データが正しければ処理区分に応じて、新マスタレコードを更新し、新マスタファイルに書き出す動作を行っていたが、ファイルクラス220ではデータが正しければ処理区分に応じてマスタレコードを更新し、マスタファイルを書き換えるという処理に変換する必要
15 がある。

上記第1～第3の対応付けに従い、抽出／変換部1600は、マッチングプログラム107で存在していたトランザクションファイルからレコードを読み込むロジックを無視し、トランザクションレコードを引数として定義する。

20 また、抽出／変換部1600は、マッチングプログラム107で存在していたマスタファイルの定義のどちらか一方を無視し（この例では新マスタファイルは無視する）マスタファイルへの読み書きは無視しなかった残る一方のファイルのみに行うようにロジックを変換する。

25 また、抽出／変換部1600は、マスタファイルを索引編成ファイルにする。

抽出／変換部 1 6 0 0 は、マスタレコードを書き出す命令を、追加（WR I T E）、更新（REWR I T E）、削除（DELE T E）にする。

5 トランザクションレコード、マスタレコードとも、ファイルクラス 2 2 0 においては一件のみ処理するので、抽出／変換部 1 6 0 0 は次のレコードを準備する繰り返しのロジックを無視する。

10 以上の方針に基づき、抽出／変換部 1 6 0 0 は変換前のマッチングプログラム 1 0 7 とソートプログラム 1 0 4 とから必要な情報を抽出、変換し、ファイルクラス 2 2 0 のソースコードを自動生成する。

15 上記抽出／変換部 1 6 0 0 による抽出、変換の詳細を図 2 6 ～図 2 8 に示す。図 2 6、図 2 7 は、マッチングプログラム 1 0 7 からファイルクラス 2 2 0 への抽出、変換の詳細を示す図である。図 2 8 は、ソートプログラム 1 0 4 からファイルクラス 2 2 0 への抽出、変換を示す図である。図 2 6 ～図 2 8 に示された位置番号は図 1 4 の対応付けに示された位置番号と一致している。このように、抽出／変換部 1 6 0 0 が自動的に必要な情報を抽出変換し、ファイルクラス 2 2 0 が生成される。

20 抽出／変換部 1 6 0 0 は、コボルプログラム 1 0 0 にはないが中間プログラム 2 0 0 で必要となる要素を新規に生成して追加する。すなわち、インターフェースクラス 2 1 0 に次の 3 つの要素を追加する。第 1 の要素は、画面からの入力終了かを判定するフラグ項目である。第 2 の要素は、画面表示メソッドや、入力チェックメソッドを呼び出すメイン手続である。第 3 の要素は画面表示メソッド
25 内の表示及び入力受付命令である。

 以上に説明したマッチングプログラム 1 0 7 のソースコードを図

29～図31に示す。また、ソートプログラム104のソースコードを図32に示す。さらに、ファイルクラス220のプログラムを図33、図34に示す。

5 プログラム中の位置情報は図14、図26及び図27及び図28との対応を明確に示すために記載しているが、実際のプログラムには不要である。

本実施の形態の変換装置A1000によれば、集中型処理に適合したコボルプログラム100のソースコードから分散型処理に適合した中間プログラム200のソースコードへの自動変換が可能となる。従って、集中型処理に用いられていたプログラム資産を分散型処理においても使用することができるため、プログラム資産の有効活用が図れる。

また、本発明の実施の形態によれば、旧来コーディングされたプログラム中の業務ロジックを再利用することができる。

15 また、プログラムの変換作業は変換装置A1000によって自動的に行われるために、人的労力を必要としないため、労力の低減を図ることができる。

また、企業などの組織において、集中型処理から分散型処理へシステムを移行する場合に、集中型処理に用いていたプログラムを分散型処理においても有効に使用できるため、新たにプログラムを開発する必要を最小限に抑えることができ、人的労力の軽減、システム構築期間の短縮化及びシステム構築費用の低減を図ることができる。

25 また、手続き型プログラムからオブジェクト指向プログラムに変換することによって、プログラム間のインターフェースをより明確にできる。

また、手続き型プログラムからオブジェクト指向プログラムに変換することによって、オブジェクト内部の仕様変更が外部に及ばないようなプログラムが可能になり、ソースコードを再利用しやすくなる。

- 5 次に、図 1 に記載された中間プログラム 2 0 0 から最終プログラム 3 0 0 を生成する変換装置 B 2 0 0 0 について説明する。この変換装置 B 2 0 0 0 により、中間プログラム 2 0 0 をよりオブジェクト指向的な、個別の役割に特化したクラスからなる最終プログラム 3 0 0 に変換することができる。

- 10 まず、中間プログラム 2 0 0 から最終プログラム 3 0 0 へプログラム変換を行う変換装置 B 2 0 0 0 の内部構成について説明する。

図 3 5 は変換装置 B 2 0 0 0 の内部構成図である。

入力部 2 1 0 0 は変換装置 A 1 0 0 0 によって変換された中間プログラム 2 0 0 を入力する。

- 15 次に、抽出／変換部 2 2 0 0 が入力部 2 1 0 0 によって入力された中間プログラム 2 0 0 から必要な情報を抽出し、変換する。出力部 2 3 0 0 は抽出／変換部 2 2 0 0 によって抽出変換されたプログラムを最終プログラム 3 0 0 として出力する。入力部 2 1 0 0、抽出／変換部 2 2 0 0、出力部 2 3 0 0 は必要に応じて記憶部 2 4 0 0 に情報を記憶することができる。たとえば、入力部 2 1 0 0 は入力した中間プログラム 2 0 0 を記憶部 2 4 0 0 に記憶することができる。なお、記憶部 2 4 0 0 は必ずしも変換装置 B 2 0 0 0 の内部に存在する必要はなく、外部記憶装置を利用してもよい。

- 25 次に、変換装置 B 2 0 0 0 により中間プログラム 2 0 0 を最終プログラム 3 0 0 に変換する場合において、各データ処理の変換の流れを説明する。

図36は、変換装置B2000によってデータ処理の方法がどのように変換されたかを示す図である。

図の左側は中間プログラム200によるオンライン一件別処理を示している。図の右側は、最終プログラム300による、中間プログラム200よりさらにオブジェクト指向的なオンライン一件別処理を示している。

左側の処理を行う中間プログラム200はインターフェースクラス210とファイルクラス220とを持つ。左側のデータの流れについては図3の説明で行なったので、ここでは省略する。

右側の最終プログラム300は5つのクラスから成り立っている。具体的にはインターフェースクラス210からビュークラス310、制御クラス320及びモデルクラス330の3つのクラスが生成される。また、ファイルクラス220からセッションクラス340及びエンティティクラス350の2つのクラスが生成される。

ビュークラス310と制御クラス320とモデルクラス330とはクライアント機器5000側のクラスである。このうち、ビュークラス310は画面表示と入力の受付を行うクラスである。モデルクラス330はデータのモデル（属性など）を管理するクラスである。制御クラス320はビュークラス310によって管理される画面と、モデルクラス330によって管理されるデータモデルの制御を行うクラスである。

セッションクラス340とエンティティクラス350とはサーバ機器6000側のクラスである。セッションクラス340はマスタレコードとトランザクションレコードの照合を行うクラスである。エンティティクラス350はセッションクラス340によって行なわれた照合の結果を記憶媒体へ書き込むことを管理するクラスであ

る。

このような各クラスの制御に従って、トランザクションレコード 303 をクライアント機器 5000 側からサーバ機器 6000 側に受け渡し、よりオブジェクト指向的なオンライン一件別処理が可能となる。

次に最終プログラム 300 を生成するために抽出／変換部 2200 が行う動作について説明する。

まず、抽出／変換部 2200 は最終プログラム 300 に予め作り込む要素を生成する。生成した要素は、図 36 に示した 5 つのクラスに対応させて 5 つのオブジェクト指向プログラムのテンプレートに記憶させておく。また、各クラスのファイル名情報を各クラスに対応するテンプレートに付けておく。

また、抽出／変換部 2200 は、各クラスにそれぞれ必要なメソッドを作り、中身は空にしておく。この時、意味情報を各メソッドに付与しておき、後で抽出変換の際に各メソッドを特定できるようにする。具体的には、ビュークラス 310 には初期化メソッドと画面表示入力メソッドを作成する。制御クラス 320 には初期化メソッドと UI メインメソッドを作成する。モデルクラス 330 には初期化メソッドと入力チェックメソッドと、画面データ受け取りメソッドを作成する。セッションクラス 340 には初期化メソッドとトランザクションチェックメソッドを作成する。エンティティクラス 350 には初期化メソッドとマスタファイル存在チェックメソッドとマッチング更新メソッドとを作成しておく。

次に、抽出／変換部 2200 が中間プログラム 200 から必要な情報を抽出変換し、最終プログラム 300 を生成する動作について、まず、インターフェースクラス 210 からビュークラス 310、

制御クラス 3 2 0 及びモデルクラス 3 3 0 を抽出、変換する動作を説明し、その後にファイルクラス 2 2 0 からセッションクラス 3 4 0 とエンティティクラス 3 5 0 とを抽出、変換する動作を説明する。

- 5 ・ インターフェースクラス 2 1 0 から 3 つのクラスへの対応付け
 まず、インターフェースクラス 2 1 0 から 3 つのクラスへの対応付けについて説明する。

抽出／変換部 2 2 0 0 は、インターフェースクラス 2 1 0 の持つ役割のうち画面表示と入力の役割部分をビュークラス 3 1 0 に振り
 10 分ける。次に、抽出／変換部 2 2 0 0 は、入力のチェックの役割部分をモデルクラス 3 3 0 に振り分ける。そして、抽出／変換部 2 2 0 0 は、これら 2 つのクラスに振り分けた各役割を呼び出す動作制御を制御クラス 3 2 0 に振り分ける。

以上に説明したインターフェースクラス 2 1 0 から 3 つのクラス
 15 への対応付けを図 3 7 に示す。また、各クラスへの具体的な抽出変換方法を図 3 8 ～図 4 0 に示す。ここで、図 3 8 ～図 4 0 中で示した位置番号は図 3 7 の位置番号に対応している。図 3 8 はインターフェースクラス 2 1 0 のプログラムからビュークラス 3 1 0 のプログラムへの具体的な抽出変換方法を示す。図 3 9 はインターフェース
 20 クラス 2 1 0 から制御クラス 3 2 0 への具体的な抽出変換方法を示す。図 4 0 はインターフェースクラス 2 1 0 からモデルクラス 3 3 0 への具体的な抽出変換方法を示す。図 3 8 に示す抽出、変換方法に基づいて抽出／変換部 2 2 0 0 により自動生成されたビュークラス 3 1 0 の具体的なプログラムのソースコードを図 4 1、図 4 2 に示す。
 25 なお、位置番号は変換前のプログラムのソースコードと変換後のプログラムのソースコードを対応付けるために記載されたものであり

、実際のプログラム中には存在する必要はない。変換前のインターフェースクラス 2 1 0 は図 2 2、図 2 3 に明示されているが、このインターフェースクラス 2 1 0 のプログラム中に記載された位置番号 (2-1) (2-2) (2-3) と図 4 1、図 4 2 に示すビュー
 5 クラス 3 1 0 のプログラム中に明示された位置番号 (2-1) (2-2) (2-3) とが対応つけられている。

図 3 9 に示す抽出、変換方法に基づいて抽出／変換部 2 2 0 0 によって自動生成された制御クラス 3 2 0 のプログラムのソースコードを図 4 3 に示す。インターフェースクラス 2 1 0 と制御クラス 3
 10 2 0 とは、これら 2 つのプログラムのソースコードに明示された位置番号 (2-1) (2-4) によって対応付けがなされている。

さらに、図 4 0 に示す抽出、変換方法に基づいて抽出／変換部 2 2 0 0 によって自動生成されたモデルクラス 3 3 0 のプログラムのソースコードを図 4 4、図 4 5 に示す。インターフェースクラス 2
 15 1 0 とモデルクラス 3 3 0 とは、位置番号 (2-1) (2-2) (2-5) (2-6) (2-7) (2-8) (2-9) によって対応付けがなされている。

- ・ファイルクラス 2 2 0 から 2 つのクラスへの対応付け

次に、ファイルクラス 2 2 0 から 2 つのクラス (3 4 0, 3 5 0)
 20) への対応付けについて説明する。抽出／変換部 2 2 0 0 は、ファイルクラス 2 2 0 の持つ役割をセッションクラス 3 4 0 及びエンティティクラス 3 5 0 の 2 つのクラスに次のように振り分ける。すなわち、ファイルクラス 2 2 0 では、マスタファイルの更新又は削除の場合には、トランザクションレコードに該当するものがマスタ
 25 ファイルに存在することを確認し、マスタファイルへの追加の場合には、トランザクションレコードに該当するものがマスタファイルに

存在しないことを確認するロジックが存在していた。抽出／変換部
2 2 0 0 は、これらのロジックをセッションクラス 3 4 0 に振り分
ける。また、抽出／変換部 2 2 0 0 は、トランザクションレコード
の処理区分に応じてマスタファイルにトランザクションレコードを
5 追加または更新または削除するロジックをエンティティクラス 3 5
0 に振り分ける。

このような対応づけによりファイルクラス 2 2 0 のプログラムの
ソースコードは変換装置 B 2 0 0 0 によってセッションクラス 3 4
0 のプログラムのソースコードとエンティティクラス 3 5 0 のプロ
10 グラムのソースコードに変換される。

上記抽出／変換部 2 2 0 0 によるファイルクラス 2 2 0 からセッ
ションクラス 3 4 0 及びエンティティクラス 3 5 0 への対応付けを
図 4 6 に示す。

また、ファイルクラス 2 2 0 のプログラムからセッションクラス
15 3 4 0 のプログラムへ変更するための具体的抽出変換方法を図 4 7
に示す。また、ファイルクラス 2 2 0 のプログラムからエンティテ
ィクラス 3 5 0 のプログラムへ変換するための具体的抽出変換方法
を図 4 8 に示す。

抽出／変換部 2 2 0 0 は、図 4 7 に示す抽出変換方法に基づいて
20 、ファイルクラス 2 2 0 のプログラムから必要なデータを抽出し、
抽出したデータを用いてセッションクラス 3 4 0 のプログラムに自
動変換する。自動変換されたセッションクラス 3 4 0 のプログラム
を図 4 9、図 5 0 に示す。

また、抽出／変換部 2 2 0 0 は、図 4 8 に示す抽出変換方法に基
づいて、ファイルクラス 2 2 0 のプログラムから必要なデータを抽
25 出し、抽出したデータを用いてエンティティクラス 3 5 0 のプログ

ラムに自動変換する。自動変換されたエンティティクラス 3 5 0 のプログラムを図 5 1 ～図 5 3 に示す。

上記生成された最終プログラム 3 0 0 の 5 つのクラスには、中間プログラム 2 0 0 には含まれない要素が存在する。そのため、その要素を新規に生成する必要がある。新規に作成すべき各クラスへの追加の要素を説明する。

この追加の要素作成は、抽出／変換部 2 2 0 0 によって行なわれる。

まず、抽出／変換部 2 2 0 0 がビュークラス 3 1 0 への新規事項追加を行う動作について図 4 1 を用いて説明する。

・初期化メソッド

初期化メソッドの内容として自己インスタンスを生成し、制御クラスの初期化メソッドを呼び出すステップを追加する。

具体的には、メソッドのデータ部に WORKING - S T O R A G E S E C T I O N を設け、自己インスタンスを参照するステップを追加する（ステップ 1 9 ～ 2 0）。メソッドの手続き部には、自己インスタンスを生成するステップ（ステップ 2 2 ～ステップ 2 3）、また自己インスタンスを引数として制御クラスの初期化メソッドを呼び出すステップ（ステップ 2 4 ～ステップ 2 5）を追加する。

次に、抽出／変換部 2 2 0 0 が制御クラス 3 2 0 へ追加する事項について図 4 3 を用いて説明する。

・初期化メソッド

初期化メソッドの内容として自己インスタンスを生成し、引数のビューインスタンスと自己インスタンスとをつなげるステップを追加する。またモデルクラス 3 3 0 の初期化メソッドを呼び出し、戻

り値のモデルインスタンスを自己インスタンスとつなげるステップを追加する。

具体的には、メソッドのデータ部にLINKAGE SECTIONを設け、引数のビューインスタンスを参照するステップを追加
 5 する（ステップ20～ステップ21）。またWORKING-STORAGE SECTIONを設け、自己インスタンスを参照する
 ステップを追加する（ステップ22～ステップ23）。メソッドの
 手続き部には、ビューインスタンスを引数として受け取ることを宣
 言するステップ（ステップ24）、自己インスタンスを生成するス
 10 テップ（ステップ25～ステップ26）、引数のビューインスタ
 ンスと自己インスタンスとをつなげるステップ（ステップ27～ステ
 ップ28）、モデルクラス330の初期化メソッドを呼び出し、戻
 り値のモデルインスタンスを自己インスタンスとつなげるステップ
 （ステップ29～ステップ31）、また制御クラス320のUIメ
 15 インメソッドを呼び出すステップ（ステップ32）を追加する。

次に、抽出／変換部2200がモデルクラス330へ追加する事項について図44、図45を用いて説明する。

・初期化メソッド

初期化メソッドの内容として自己インスタンスを生成し、引数の
 20 ビューインスタンスと自己インスタンスとをつなげ、自己インスタ
 ンスを戻り値として設定するステップを追加する。

具体的には、メソッドのデータ部にLINKAGE SECTIONを設け、引数のビューインスタンスを参照するステップと戻り
 値の自己インスタンスを参照するステップを追加する（ステップ2
 25 0～ステップ22）。またWORKING-STORAGE SECTIONを設け、自己インスタンスを参照するステップを追加す

- る（ステップ23～ステップ24）。メソッドの手続き部には、ビューインスタンスを引数として受取り自己インスタンスを戻り値として返すことを宣言するステップ（ステップ26～ステップ27）、自己インスタンスを生成するステップ（ステップ28～ステップ29）、引数のビューインスタンスと自己インスタンスとをつなげるステップ（ステップ30～ステップ31）、また自己インスタンスを戻り値として設定するステップ（ステップ32）を追加する。

次に、抽出／変換部2200がセッションクラス340へ追加する事項について図49、図50を用いて説明する。

10 ・初期化メソッド

初期化メソッドの内容として自己インスタンスを生成し、エンティティクラス350の初期化メソッドを呼び出して戻り値のエンティティインスタンスを自己インスタンスにつなげ、自己インスタンスを戻り値として設定するステップを追加する。

- 15 具体的には、メソッドのデータ部にLINKAGE SECTIONを設け、戻り値の自己インスタンスを参照するステップを追加する（ステップ18～ステップ19）。またWORKING-STORAGE SECTIONを設け、自己インスタンスを参照するステップを追加する（ステップ20～ステップ21）。メソッドの手続き部には、自己インスタンスを戻り値として返すことを宣言するステップ（ステップ22）、自己インスタンスを生成するステップ（ステップ23～ステップ24）、エンティティクラスの初期化メソッドを呼び出し、戻り値のエンティティインスタンスを自己インスタンスとつなげるステップ（ステップ25～ステップ26）、
- 20 また自己インスタンスを戻り値として設定するステップ（ステップ27）を追加する。
- 25

・トランザクションチェックメソッド

トランザクションチェックメソッドにはエンティティクラス 350 のマスタレコード存在チェックメソッドを呼び出し、結果が正しければエンティティクラス 350 のマッチング更新メソッドを呼び出すステップを追加する。

具体的には、メソッドのデータ部に WORKING-STORAGE SECTION を設け、エンティティクラス 350 のマスタレコード存在チェックメソッドからの戻り値を格納するステップを追加する（ステップ 61～ステップ 64）。メソッドの手続き部には、エンティティクラス 350 のマスタレコード存在チェックメソッドを呼び出すステップ（ステップ 67～ステップ 68）、またこの呼び出しの結果が正しければエンティティクラス 350 のマッチング更新メソッドを呼び出すステップ（ステップ 69～ステップ 82）を追加する。

最後に抽出／変換部 2200 がエンティティクラス 350 のプログラムへ追加する事項について図 51、図 52 を用いて説明する。

・初期化メソッド

初期化メソッドの内容として、自己インスタンスを生成し、これを戻り値として返すステップを追加する。

具体的にはメソッドのデータ部に LINKAGE SECTION を設け、戻り値の自己インスタンスを参照するステップを追加する（ステップ 17～ステップ 18）。また WORKING-STORAGE SECTION を設け、自己インスタンスを参照するステップを追加する（ステップ 19～ステップ 20）。メソッドの手続き部には、戻り値として自己インスタンスを返すことを宣言するステップ（ステップ 21）、自己インスタンスを生成するステップ

(ステップ22～ステップ23)、戻り値に自己インスタンスを設定するステップ(ステップ24)を追加する。

・マスタレコード存在チェックメソッド

マスタレコード存在チェックメソッドの内容として、レコードキーを引数に受取り、そのキーでマスタファイルを読み込み、その結果をマスタレコード存在フラグに格納して戻り値として返すステップを追加する。

具体的には、メソッドのデータ部にLINKAGE SECTIONを設け、引数のレコードキーを格納するステップと戻り値のマスタレコード存在フラグを格納するステップを追加する(ステップ62～ステップ66)。メソッドの手続き部には、レコードキーを引数として受取りマスタレコード存在フラグを戻り値として返すことを宣言するステップ(ステップ67～ステップ68)、マスタファイルを読み込む準備をするステップ(ステップ70～ステップ71)、またマスタファイルを読み込みその結果を戻り値として設定するステップ(ステップ72～ステップ77)を追加する。

このようにして変換装置B2000は中間プログラム200のソースコードから最終プログラム300のソースコードを自動変換することにより、よりオブジェクト指向性の高いプログラムを生成することができる。このようにオブジェクト指向性の高いプログラムに変換することによって、オブジェクト内部の仕様変更が外部に及ばないようなプログラムが可能になり、ソースコードを再利用しやすくなる。よって、過去において作成されたプログラム資産を分散型処理でさらに有効活用することが可能となる。

また、WEBやXMLなどに容易に連携することができるため、中間プログラム200よりも現代社会のネットワークシステムによ

り合致したプログラム構造と言える。従って、このようにして、従来のプログラムから自動的に人的労力をかけずに自動変換された最終プログラム 300 によれば、現在主流となっている分散型処理のインフラをより有効に活用することができるアプリケーションプログラムとして再利用することが可能となる。

実施の形態 2.

次に実施の形態 2 について説明する。本実施の形態では、実施の形態 1 のように中間プログラム 200 を生成するステップを設けず、コボルプログラム 100 のソースコードから直接最終プログラム 300 のソースコードを抽出、変換する形態である。

図 5 4 は、本実施の形態の概念図である。

本実施の形態では、変換装置 C 3000 がコボルプログラム 100 のソースコードを直接最終プログラム 300 のソースコードに変換している。このように、変換装置 C 3000 が自動的に直接最終プログラム変換を行うことにより、中間プログラム 200 を生成する段階を設けずに、集中型処理から WEB や XML などに連携できる分散型処理に適したプログラムを短期間に容易に取得することができる。

変換装置 C 3000 の構成及び動作について説明する。図 5 5 は、変換装置 C 3000 の内部構成図である。

実施の形態 1 の変換装置 A 1000 の内部構成を示す図 4 と比べ、内部構成自体は同一である。ただし、出力部 1700 が出力するプログラムが最終プログラム 300 であることと、抽出／変換部 3100 の動作が一部異なっている。即ち、変換装置 A 1000 の抽出／変換部 1600 では中間プログラム 200 を生成するためにデ

データの抽出変換を行なっていたが、本実施の形態の抽出／変換部 3 1 0 0 では、最終プログラム 3 0 0 を生成するためにデータの抽出及び変換を行なっている。

5 入力部 1 1 0 0 はコボルプログラム 1 0 0 を入力し、記憶部 1 8 0 0 に記憶する。分割部 1 2 0 0 はコボルプログラム 1 0 0 をまとまりある複数のプログラムに分解し、構文解析部 1 3 0 0 は分解されたそれぞれのプログラムの構文を解析する。プログラム判断部 1 4 0 0 は各プログラムの役割を判断し、節判断部 1 5 0 0 は各プログラム中の節の内容、役割を判断する。

10 これらの動作を終えた後、抽出／変換部 3 1 0 0 は最終プログラム 3 0 0 を生成するためのデータの抽出及び変換を行ない、その結果、生成された最終プログラム 3 0 0 は出力部 1 7 0 0 によって出力される。この場合、記憶部 1 8 0 0 は必要に応じてデータを記憶する領域として利用することができ、また 3 0 0 0 の内部に記憶部 1 8 0 0 が存在しなくても、外部記憶装置に記憶させてもよい。

20 このように、本実施の形態では、コボルプログラム 1 0 0 から中間プログラム 2 0 0 を出力することなく、直接最終プログラム 3 0 0 を出力するため、変換処理を高速に行え、WEBやXMLなどに連携できる分散型処理に適したプログラムを短期間に容易に取得することができる。

25 また、最終プログラム 3 0 0 を中間プログラム 2 0 0 と比較すると機能自体は変わらないが、最終プログラム 3 0 0 ではプログラムの部品化が進むため、既存部品に差分だけを付け加えて必要な動作を実行するプログラムを容易に作成できる。このため、さらに資産価値が高いプログラムを取得することができる。

また、最終プログラム 3 0 0 を中間プログラム 2 0 0 を J a v a

（登録商標）言語やC++言語等の言語を用いて記述することが可能である。

5 なお、上記全ての実施の形態では、コボルプログラムを基に変換プログラムを生成したが、変換前のプログラムはコボルプログラムに限る必要はなく、バッチ処理をする構造化されたプログラムであればよい。従って、構造化されたプログラムであれば、上記変換装置により、分散型処理に適合したプログラムに変換することができる。

10 図56は、変換装置A、変換装置B、変換装置Cのコンピュータ基本構成図である。

図56において、プログラムを実行するCPU（Central Processing Unit）40は、バス38を介してモニタ41、キーボード42、マウス43、通信ボード44、磁気ディスク装置46等と接続されている。

15 磁気ディスク装置46には、オペレーティングシステム（OS）47、プログラム群49、ファイル群50が記憶されている。ただし、プログラム群49、ファイル群50が一体となってオブジェクト指向のプログラム群49を形成する形態も一実施の形態として考えられる。

20 プログラム群49は、CPU40、OS47により実行される。

上記各実施の形態では、変換装置A、変換装置B、変換装置Cは、通信ボード44の機能を使用して、各種ネットワークを経由して接続された機器と通信を行う。

25 以上に記載した「格納する」、「記憶する」という用語は、記録媒体に保存することを意味する。

すべての実施の形態では、各構成要素の各動作はお互いに関連し

ており、各構成要素の動作は、上記に示された動作の関連を考慮しながら、一連の動作として置き換えることができる。そして、このように置き換えることにより、変換装置の実施形態を変換方法の発明の実施形態とすることができる。

- 5 また、上記各構成要素の動作を、各構成要素の処理と置き換えることにより、変換プログラムの実施の形態とすることができる。

 また、変換プログラムをコンピュータ読み取り可能な記録媒体に記憶させることで、変換プログラムを記録したコンピュータ読み取り可能な記録媒体の実施の形態とすることができる。

- 10 変換プログラムの実施の形態及び変換プログラムを記録したコンピュータ読み取り可能な記録媒体の実施の形態は、すべてコンピュータで動作可能なプログラムにより構成することができる。

- また、変換プログラムの実施の形態および変換プログラムを記録したコンピュータ読み取り可能な記録媒体の実施の形態における各処理はプログラムで実行されるが、このプログラムは、記録装置に記録
15 されていて、記録装置から中央処理装置（CPU）に読み込まれ、中央処理装置によって、プログラムに記述された動作が実行されることになる。

- また、各実施の形態に記載されたソフトウェアやプログラムは、
20 ROM（READ ONLY MEMORY）に記憶されたファームウェアで実現されていても構わない。あるいは、ソフトウェアとファームウェアとハードウェアとの組み合わせで前述したプログラムの各機能を実現しても構わない。

- 25 実施例.

 以下に、一括処理方式のコボルプログラム（レガシープログラム

）の変換法の実務プログラムへの適用実施例について説明する。この実施例では、前述した実施の形態と同一又は相当する部分に対しては、前述した実施の形態と同一符号を用いて説明する。

前述した実施の形態では、一括処理方式のCOBOLのレガシー
5 プログラムを、クライアント／サーバやWeb技術を利用した新しい形のプログラムに自動変換する手法を説明した。すなわち、前述した実施の形態では、一括処理を行うCOBOLプログラムを対象とし、一件別処理のオブジェクト指向COBOLプログラムに自動的に変換するアルゴリズムを説明した。これを実際に企業で使用さ
10 れているCOBOLプログラムに適用し、前述した実施の形態で説明した変換法がある程度有効であることを確かめたので、以下に、この実施例の変換法と、実験に用いたCOBOLプログラムの内容、実験結果について述べる。

1. はじめに

15 本実験の目的は、集中型、オフライン一括処理を行う手続き型のプログラムを、オンライン一件別のオブジェクト指向プログラムに自動的に変換し、さらにクライアント／サーバ環境で動作するWebプログラムに自動的に変換する方法の確立である（図57）。

この実験例では、一括処理のCOBOLプログラムソースコード
20 を元に、クライアント側とサーバ側に分かれ、クライアント側ではMVC（Model, View, Controller）パターンに対応し、サーバ側ではEJB（Enterprise Java Beans（商標））のトランザクション処理モデルに対応した構造のオブジェクト指向COBOL（OO-COBOL）プログラムソ
25 ースコードを生成する方法の実装を試みた。

この実験例では、変換の方針、規則などを検討するにあたっては

、COBOLシステムの資料（日立製作所：COBOL 85プログラミング、6190-3-724（1995））に載っているサンプルのプログラムを使用した。そして、この変換方法が企業で実際に使用されているCOBOLプログラムに対して適用できるか確かめることにした。今回システムインテグレータ1社の協力によりその機会を得た。以下、変換のアルゴリズムと実務プログラムを使用した実験について述べる。

2. サンプルを元にしたプログラム変換のアルゴリズム

プログラムの変換は、2段階に分けて行う（図58、図59）。まず、意味付与などの前処理をした後、手続きの流れを一括処理から一件別処理に変えながらクライアント側とサーバ側の2種類のクラス・プログラム210、220に変換する（図58）。この変換プログラムを、ここでは、変換1プログラムと呼ぶことにする（あるいは、単に、変換1ともいう）。また、変換1プログラムを実行するハードウェアとソフトウェアの環境を第1の変換部という。

次いでこの2種類のクラス・プログラム210、220を、クライアント側はMVCの3種類のクラス・プログラム310、320、330に変換し、またサーバ側はSessionとEntityの2種類のクラス・プログラム340、350に変換する。この変換プログラムを、ここでは、変換2プログラムと呼ぶことにする（あるいは、単に、変換2ともいう）。また、変換2プログラムを実行するハードウェアとソフトウェアの環境を第2の変換部という。第1の変換部と第2の変換部とは同一の計算機で実現できる。

（2-1）変換1プログラムによるプログラムへの意味の自動付与
意味の付与とは、プログラムの役割や、プログラム内のデータの宣言・ひと続きの手続き等の要素が一連の処理の中で果たす役割を

検出し、プログラムやその要素に対して役割を表すラベルを付与することである。

意味の付与は、次のことを行うための準備として必要である。

5 (1) COBOL プログラムをフレームワークに対応づけて変換するための、プログラムの役割の判断とそのデータや手続きの役割の判断。

(2) プログラムの手続きを一括処理から一件別処理に変えるための、繰り返しなど一括処理のロジックの検出。

10 予め個々のプログラムに意味のラベルを付与し、さらに、データや手続きに意味のラベルを付与しておき、変換を行う際に要素を抽出するための手がかりとする。

プログラムの役割判断やデータの役割判断、特定のロジックなどの要素の判定は、構文のパターンだけでは判別できないため、意味の付与によって補う。

15 この意味の付与は、プログラムソースコード内にあるデータの定義を参照し、マスタファイルの定義であるとか、トランザクションファイルの定義であるとかを自動的に判定し、判定結果をプログラムソースコード内に自動で付与する前処理プログラムルーチンにより実現できる。この意味の付与は、変換 1 プログラムの前段処理で
20 実行する。

(2-2) 変換 1 プログラムによる変換

変換 1 プログラムでは、図 5 9 に示した 4 種類の COBOL プログラム 1 0 2、1 0 4、1 0 7、1 0 9 を、クライアント側とサーバ側の 2 種類の OO-COBOL クラス 2 1 0、2 2 0 に変換する
25 。クライアント側クラス 2 1 0 は、画面の入出力、入力データのチェックなど、ユーザ・インタフェースの機能を持つ。またサーバ側

クラス 2 2 0 は、トランザクションのマッチング・更新の機能を持つ。

変換では、予め 2 つのクラスの構造と、必要なメソッドのスケルトンを持つテンプレートを用意し、そこに COBOL プログラムから情報を抽出・変換してあてはめる方式を採る。

a. クライアント側クラス 2 1 0 の生成

クライアント側クラス 2 1 0 は、画面などから入力データを受け付けてトランザクション・レコードに格納し、チェック処理を行い、正しいデータはチェック済トランザクションに格納してサーバ側クラス 2 2 0 に渡す。この処理を行う属性とメソッドをクラス内に生成するために、入力・チェックプログラム 1 0 2 および結果出力プログラム 1 0 9 から要素の抽出・変換を行う（図 6 0）。

b. サーバ側クラス 2 2 0 の生成

サーバ側クラス 2 2 0 は、クライアント側クラス 2 1 0 からトランザクション・レコードを受け取り、マスタ・レコードとのマッチングを行った結果が正しければ、追加、更新、削除などの処理を行う。ソートプログラム 1 0 4 およびマッチング・更新プログラム 1 0 7 から要素の抽出・変換を行う（図 6 1）。

（2-3）変換 2 プログラム

変換 2 プログラムでは、変換 1 プログラムによって生成したクライアント側クラス 2 1 0 を MVC パターンに対応付けて 3 種類のクラス 3 1 0、3 2 0、3 3 0 に分け、サーバ側クラス 2 2 0 を EJB モデルに対応付けて 2 種類のクラス 3 4 0、3 5 0 に分ける（図 6 2、図 6 3）。

変換 1 プログラムと同様に、変換 2 プログラムは予め 5 つのクラスの骨格を持つテンプレートを用意しておき、そこに変換 1 プログ

ラムによるOO-COBOLクラス210、220から情報を抽出・変換してあてはめる。

a. クライアント側クラス210のMVCクラスへの分解

5 クライアント側クラス210で行っていた処理のうち、画面表示処理と入力の受付け処理をViewクラスに、チェック処理をModelクラスに、これら2つのクラスの制御の役割をControllerクラスにふり分ける（図62）。

b. サーバ側クラス220のSessionクラス340、Entityクラスへ350の分解

10 サーバ側クラス220で行っていた処理のうち、トランザクション・レコードとマスタ・レコードとのマッチング処理をSessionクラス340に、追加、更新、削除などの更新処理をEntityクラス350にふり分ける（図63）。

15 以上がこの実施例の基礎となる基本的変換法である。この変換法について、COBOLを扱うシステムインテグレータ1社で実際に運用されているプログラムを使用して評価を行う機会を得た。実験内容について以下の「3. 実務プログラムへの適用実験」で述べる。

3. 実務プログラムへの適用実験

20 3つのジョブ、計17本のプログラムの入出力や手続きの特徴を分析し、次の3種類の一括処理に分けられることがわかった。

（種類1）トランザクションを入力し、キーの変わり目を判定して集計を出力する処理。

25 （種類2）トランザクションを入力し、新しい項目を追加するなどレコードの形式を変えて出力する処理。

（種類3）うえの2種類にあてはまらない処理。

上記計 17 本のプログラムの中から上記 (1)、(2)、(3) にそれぞれに該当する典型的なプログラムを各 1 本、計 3 つの一括処理プログラムを抽出した。以下に実験の方針、個々のプログラムの変換の実際について述べる。

5 (3-1) 実験の方針

実験の目的と手順を次のように設定して行った。

(1) 実験の目的

サンプルのプログラムを用いて、変換アルゴリズムが、実務のプログラムにもあてはまるか検証する。すなわち、

10 (検証目的 1) : 命名規則からプログラム中のデータの役割が判断できるか。

(検証目的 2) : 手続きの節、段落の呼出し関係から、個々の要素の役割を導出判断できるか。

15 (検証目的 3) : データの定義を移動する操作、繰返しのロジックを除去する操作、などの要素に対する操作を行えるか。

主にこれらの前提が実務のプログラムにもあてはまるか、変換を試行して確かめる。

(2) 実験の手順

a. 変換の方針の決定

20 最終的にどのような O O - C O B O L プログラムに変換するか検討する。

①データの入出力の変更、

②手続きの一括処理から一件別処理への変更、

③必要に応じて、元のプログラムのデータを組み合わせた新しいフ

25 ァイルの導入、

などの変換の方針を決める。

b. 変換後のプログラムの構築

決定した方針に従って、変換して出来上がったOO-COBOLプログラムを人手で組む。

c. 変換規則の検討

- 5 元のプログラムと、上記「b. 変換後のプログラムの構築」で構築した変換後のプログラムとを比較し、元のプログラム内の要素に対する変換の規則を検討する。

- 10 上記「a. 変換の方針の決定」、「b. 変換後のプログラムの構築」、「c. 変換規則の検討」は、変換1プログラムと変換2プログラムとを作成するために行うものであり、変換1プログラムと変換2プログラムが一旦作成された場合は、実行する必要はない。ただし、上記「a. 変換の方針の決定」、「b. 変換後のプログラムの構築」、「c. 変換規則の検討」は、3つの一括処理プログラムを対象にそれぞれ個々に行うものであり、3つの一括処理プログラムそれぞれに対して、変換1プログラムと変換2プログラムとがそれぞれ3種類作成されることになる。3種類の変換1プログラムと変換2プログラムが一旦作成された後は、その種類に属する集中型、オフライン一括処理を行う手続き型のプログラムソースコードとデータの命名規則と手続きの組み方の規則を入力データとして変換
- 15 1プログラムを実行することにより、オンライン一件別のオブジェクト指向プログラムに自動的に変換でき、さらに、変換2プログラムを実行することにより、クライアント/サーバ環境で動作するWebプログラムに自動的に変換できる。
- 20

- 25 変換のプロセス中で最も複雑な操作を行っているのは、変換1プログラムである。変換1は、元のプログラムの入出力を変更し手続きを一括処理から一件別処理に変更する段階である。そこでこの変

換 1 の段階について特に詳細に述べる。

(3-2) 一件別処理への変更

データと手続きの特徴に基づき以下の 3 つの一括処理プログラムを抽出した。

5 (1) 第 1 の一括処理プログラム：「集計・出力プログラム 9 0 1」

(2) 第 2 の一括処理プログラム：「レコード形式変換プログラム 9 0 2」

10 (3) 第 3 の一括処理プログラム：「出力指示データ作成プログラム 9 0 3」

抽出した 3 つの一括処理プログラムを、次のように一件別処理に変更し、一つのメソッドとして利用する。以下に、3 つの一括処理プログラムの各プログラムについて、

- a. 現行の処理内容、
 - 15 b. 変換後の処理内容、
 - c. 手続きの変換、
 - d. それらに基づく変換規則
- を述べる。

(1) 第 1 の一括処理プログラム「集計・出力プログラム 9 0 1」

20 a. 現行の処理内容

トランザクションをファイルから順次読み込み、部門、店の変更（ブレイク）があれば、小計合計を計算・出力する一括処理を行う（図 6 4）。このときマスタ DB から文字情報を取得して印字する。

25 b. 変更後の処理内容

トランザクションをレコードとして一件別に受け取り、一件ごと

に随時に小計合計を計算する。計算結果は新しく作る集計ファイル（索引編成）に累積していく。照会するときは、この集計ファイルからデータを出力する（図 6 5）。

c. 手続きの変換

5 現行の一括処理の手続きの流れは、次のとおりである（図 6 6）

。

①トランザクションファイルから 1 レコード読む。

②明細の数値を各小計に加算する。

③小ブレイクならば、小計の合計を計算し、各小計と共に出力する

10 。

④大ブレイクならば、全合計を計算し、各合計と共に出力する。

⑤トランザクションファイルが終わるまで繰り返す。

この手続きを、上記「b. 変更後の処理内容」の方針に従って次の一件別の流れに変換する（図 6 7）。

15 ① 1 レコードが引数として入力される。

②明細の数値を各小計に加算する。

③小計の合計を計算し、集計ファイルの該当する小計レコードを更新する。

20 ④全合計を計算し、集計ファイルの該当する合計レコードを更新する。

d. 変換規則の検討

プログラムに以下のような変換を施す。

①一連の手続きをループする繰返しの指定を除去する。

25 ②トランザクション・レコードの定義を、引数にするためにデータ部連絡節に移動する。

③ブレイクの判定を除去し、毎回ブレイク処理を行うようにする（

図 6 8)。すなわち、I F クローズによる、部門、店の変更（ブレイク）判定を削除して、トランザクション・レコード 1 件ごとに、小計合計を計算する P E R F O R M 文を無条件に実行する。

- ④リスト出力の部分を、集計ファイルの該当レコードの読み込み、集計項目の加算、ファイルの更新、に変わる。

（2）第 2 の一括処理プログラム「レコード形式変換プログラム 9 0 2」

a. 現行の処理内容

- トランザクションをファイルから順次読み込み、他のレコード形式に変換してファイルに書き出す一括処理を行う。

b. 変更後の処理内容

トランザクションを一件別にレコードの形式で受け取り、他の形式のレコードに変換し、次の処理への引数として送り出す（図 6 9）。

c. 手続きの変換

現行の一括処理の手続きの流れは、次のとおりである（図 7 0 の「（1）現行」）。

- ①トランザクションファイルから 1 レコード読む。
- ②レコードの各項目の値を、新しいレコードに転記する。
- ③新しいレコードを新ファイルに書き出す。
- ④トランザクションファイルが終わるまで繰り返す。

この手続きを、上記「b. 変更後の処理内容」の方針に従って次の一件別の流れに変換する（図 7 0 の「（2）変換後」）。

- ① 1 レコードが引数として入力される。
- ②レコードの各項目の値を、新しいレコードに転記する。
- ③新しいレコードを次の処理を行うメソッドの引数として送り出す

d. 変換規則の検討

プログラムに以下のような変換を施す。

- ①一連の手続きをループする繰返しの指定を除去する（図 7 1）。
 - 5 すなわち、UNTILクローズによるループ処理を削除し、PERFORM文のみにする。
 - ②トランザクション・レコードの定義を、引数にするためにデータ部連絡節に移動する。
 - ③新しいレコードを書き出す部分を、次の処理を行うメソッドの呼出しに変える。

（3）第3の一括処理プログラム「出力指示データ作成プログラム 903」

①. 現行の処理内容

- 15 トランザクションをファイルから順次読み込み、同じIDに属する下位のIDを並べて出力指示データとして書き出す一括処理を行う（図 7 2）。

②. 変更後の処理内容

- 20 トランザクションを一件別にレコードの形式で受け取る。出力指示データは索引ファイルに格納し、トランザクションごとに随時更新を行う（図 7 3）。

③. 手続きの変換

現行の一括処理の手続きの流れは、次のとおりである（図 7 4 の「（1）現行」）。

- ①トランザクションファイルから1レコード読む。
- 25 ②ソートファイルに書き出す。
- ③全て書き出した後、ソートを行う。

- ④ソートファイルから1レコード読む。
- ⑤出力レコード内の配列構造に指示データを転記する。
- ⑥出力レコードを書き出す。
- ⑦ファイルが終わるまで繰り返す。

5 この手続きを、上記「b. 変更後の処理内容」の方針に従って次の一件別の流れに変換する（図74の「（2）変換後」）。

- ①1レコードが引数として入力される。
- ②出力レコード内の配列構造に指示データを転記する。
- ③出力指示データのファイルから、該当するレコードを更新する。

10 d. 変換規則の検討

プログラムに以下のような変換を施す。

- ①ソート用レコードの定義を、引数にするためにデータ部連絡節に移動する。
- ②ソートの命令文からソート前手続きとソート後手続きの呼び出し指定を抽出し、PERFORM文による呼び出しに置き換えるほか、ソートに関する命令を除去する。
- ③出力レコードを書き出す部分を、出力指示データのファイルに対する更新に置き換える。

20 以上のように、本変換法を適用した変換の方針の決定、手続きの変換、変換規則の決定を行うことができた。

（3-3）クライアント／サーバ（C／S）のクラスへの分解

一括処理のプログラム901から一件別処理のメソッド901aの形に変換したプロダクトのうち、計算、転記を行ってデータの内容を変更する部分をクライアント側クラス210に分け、最終的な結果をファイルに読み書きする部分をサーバ側クラス220に分ける（図67から図75へ変換する）。一件別のメソッドへの変換結

25

果を元にして、クライアント／サーバ（C／S）のクラスへの分解を自動的に行う。サーバ側クラス 2 2 0 の読み書きのロジックは新規のものなので、テンプレートにデータ名等をあてはめて C／S のクラスを生成する。

5 (3-4) C／S のクラスから 5 つのクラスへの分解

クライアント側クラス 2 1 0 にあったロジックのほとんどは Model クラス 3 3 0 に、サーバ側クラス 2 2 0 にあったロジックは Entity クラス 3 5 0 に入る。他のクラスは、データをそのまま受け渡ししたり、メソッドの呼び出し制御を行う（図 7 5 から図 7 6 へ変換する）。C／S のクラスへの分解と同様に、C／S のクラスから自動的に 5 つのクラス 3 1 0、3 2 0、3 3 0、3 4 0、3 5 0 を生成する。また 5 つのクラスを生成する際にテンプレートを用いる点も C／S のクラスの生成と同様である。

4. 考察

15 a. アルゴリズムの実用性

実験の目的にあげた、

（検証目的 1）：プログラム中のデータの役割の判断、

（検証目的 2）：手続きの節、手続きの呼出し関係からの個々の要素の役割の導出判断、

20 （検証目的 3）：プログラム中の要素に対する操作、

などが行え、本変換法が実務のプログラムにも適用できることを確かめた。企業ではコーディング規約を定めてプログラミングを行っていることが多く、同種の複数のプログラムに対しこれらのアルゴリズムを適用できる可能性はある。

25 一方、変換を行う前にまず人手で変換の方針を検討しなければならず、その手間がかかる点が本変換法の課題である。

b. 変換による量的な変化

変換のプロセスで、元のプログラムの内容が最も変化するのが、一括処理のプログラムから一件別のメソッドの形式に変える段階である。図 7 7 に、この段階で変化する行数がプログラムに占める割合を示す。形式変換プログラムの変化の割合が大きいのは、処理が単純で行数が少ないのに対し、項目数の多いレコードを移動していることが原因である。

また、元のプログラムソースコードから C / S クラスに変換すると、プログラムソースコードの行数は元のプログラムの約 1.4 倍になり、最終的な 5 つのクラスのプログラムソースコードの行数は元のプログラムの 1.6 倍になる。

5. おわりに

以上のように、この実施例では、レガシープログラムの再利用を目的とした変換法について、実務で適用できる可能性があることを示した。

産業上の利用可能性

本発明の実施の形態によれば、旧来のプログラム資産を分散システムに対応するプログラム資産に変換することができる。

また、本発明の実施の形態によれば、旧来コーディングされたプログラム中の業務ロジックを再利用することができる。

また、本発明の実施の形態によれば、旧来のプログラム資産を利用して、新しいシステムを構築することができる。

また、本発明の実施の形態によれば、旧来のプログラム資産から中間プログラムへ変換することなく直接最終プログラムへ変換することができる。

請求の範囲

1. バッチ処理をするプログラムをソースコードの形式で記憶する記憶部と、

- 5 上記記憶部が記憶したプログラムのソースコードを1以上のまとまりある処理に区切り、区切った処理を節として節毎の役割を各節の意味情報として判断する節判断部と、

- 10 上記節判断部が判断した各節の意味情報に基づいて上記記憶部が記憶したプログラムのソースコードからソースコード変換のための変換情報を抽出し、抽出した変換情報に基づいてプログラムのソースコードをクライアント機器用の変換結果プログラムのソースコードとサーバ機器用の変換結果プログラムのソースコードとの2つからなる変換結果プログラムのソースコードに変換する抽出／変換部とを備える変換装置。

- 15 2. 上記抽出／変換部は、上記2つの変換結果プログラムのソースコードを、オブジェクト指向プログラムのソースコードに変換する請求項1に記載された変換装置。

- 20 3. 上記抽出／変換部は、所定のデータ構造と手続きをもつ複数のクラスに対応した複数のオブジェクト指向プログラムのテンプレートを生成し、上記2つの変換結果プログラムのソースコードから所定のデータ構造と手続きからなる情報を複数抽出し、抽出した各情報をテンプレートの対応する部分に適用することによって上記2つの変換結果プログラムのソースコードを複数のオブジェクト指向プログラムのソースコードに変換する請求項2に記載された変換装置。

- 25 4. バッチ処理をするプログラムをソースコードの形式

で記憶する記憶部と、

上記記憶部が記憶したプログラムのソースコードを 1 以上のまとまりある処理に区切り、区切った処理を節として節毎の役割を各節の意味情報として判断する節判断部と、

- 5 所定のデータ構造と手続きをもつ複数のクラスに対応した複数のオブジェクト指向プログラムのテンプレートを生成し、上記節判断部が判断した各節の意味情報に基づいて上記記憶部に記憶されたバッチ処理をするプログラムのソースコードから所定のデータ構造と手続きからなる情報を複数抽出し、抽出した各情報をテンプレートの
- 10 対応する部分に適用することによって上記記憶部が記憶したプログラムのソースコードを複数のオブジェクト指向プログラムのソースコードに変換する変換装置。

5. 上記変換装置は、さらに、

- 15 上記記憶部が記憶したプログラムのソースコードの役割をプログラムの意味情報として判断するプログラム判断部を備え、

上記抽出／変換部は、上記プログラム判断部が判断したプログラムの意味情報と上記節判断部が判断した各節の意味情報とに基づいてプログラムのソースコードからプログラムのソースコードを変換するための変換情報を抽出する請求項 1 に記載された変換装置。

- 20 6. 上記変換装置は、さらに、

上記記憶部が記憶したプログラムを構文解析する構文解析部を備え、

- 25 上記節判断部は、上記構文解析部によって構文解析されたプログラムに含まれる各節の意味情報を判断する請求項 1 に記載された変換装置。

7. 上記変換装置は、バッチ処理をするコボルプログラ

ムのソースコードを変換する請求項 1 に記載された変換装置。

8. バッチ処理をするプログラムをソースコードの形式で記憶し、

5 上記記憶したプログラムのソースコードを 1 以上のまとまりある処理に区切り、区切った処理を節として節毎の役割を各節の意味情報として判断し、

10 上記判断した各節の意味情報に基づいて上記記憶したプログラムのソースコードからソースコード変換のための変換情報を抽出し、抽出した変換情報に基づいてプログラムのソースコードをクライアント機器用の変換結果プログラムのソースコードとサーバ機器用の変換結果プログラムのソースコードとの 2 つからなる変換結果プログラムのソースコードに変換する変換方法。

9. バッチ処理をするプログラムをソースコードの形式で記憶する処理と、

15 上記記憶したプログラムのソースコードを 1 以上のまとまりある処理に区切り、区切った処理を節として節毎の役割を各節の意味情報として判断する処理と、

20 上記判断した各節の意味情報に基づいて上記記憶したプログラムのソースコードからソースコード変換のための変換情報を抽出し、抽出した変換情報に基づいてプログラムのソースコードをクライアント機器用の変換結果プログラムのソースコードとサーバ機器用の変換結果プログラムのソースコードとの 2 つからなる変換結果プログラムのソースコードに変換する処理とをコンピュータに実行させる変換プログラム。

25 10. バッチ処理をするプログラムをソースコードの形式で記憶する処理と、

上記記憶したプログラムのソースコードを1以上のまとまりある処理に区切り、区切った処理を節として節毎の役割を各節の意味情報として判断する処理と、

上記判断した各節の意味情報に基づいて上記記憶したプログラムのソースコードからソースコード変換のための変換情報を抽出し、
抽出した変換情報に基づいてプログラムのソースコードをクライアント機器用の変換結果プログラムのソースコードとサーバ機器用の変換結果プログラムのソースコードとの2つからなる変換結果プログラムのソースコードに変換する処理とをコンピュータに実行させるための変換プログラムを記録したコンピュータ読み取り可能な記録媒体。

1 1. オフライン一括処理を行う手続き型のプログラムソースコードを入力データとして、このプログラムソースコードをオンライン一件別処理のプログラムに変換する第1の変換部と、この
オンライン一件別処理のプログラムをクライアント／サーバ環境で動作するWebプログラムに変換する第2の変換部とを備えたことを特徴とする変換装置。

1 2. 上記第1の変換部は、オフライン一括処理を行うプログラムソースコードとデータの命名規則と手続きの組み方の規則
を入力データとして、上記プログラムソースコードをオンライン一件別処理を行うクライアント側クラスとサーバ側クラスの2種類のクラス・プログラムに変換し、

上記第2の変換部は、この2種類のクラス・プログラムを入力し、オブジェクト指向プログラムのソースコードを生成することを特徴とする請求項1 1記載の変換装置。

1 3. 上記第2の変換部は、クライアント側クラスをモデ

ルクラスとビュークラスとコントローラクラスとの3種類のクラス・プログラムに変換し、またサーバ側クラスをセッションクラスとエンティティクラスとの2種類のクラス・プログラムに変換することを特徴とする請求項12記載の変換装置。

- 5 14. 上記第1の変換部は、上記プログラムのソースコード内にあるデータの定義を参照し、マスタファイルの定義とトランザクションファイルの定義とを判定して、プログラムの役割とそのプログラム内の要素が一連の処理の中で果たす役割とを検出し、プログラムの役割とその要素が一連の処理の中で果たす役割を表すラ
- 10 ベルを付与する意味付与の前処理を備えたことを特徴とする請求項12記載の変換装置。

 15. 上記第1の変換部は変換1プログラムを実行し、上記第2の変換部は変換2プログラムを実行し、

 オフライン一括処理を行うプログラムは種類分けされ、

- 15 上記変換1プログラムと上記変換2プログラムは、オフライン一括処理を行うプログラムの各種類ごとに対応して作成され、

 上記第1の変換部と上記第2の変換部とは、入力したオフライン一括処理を行うプログラムの種類に対応して作成された変換1プログラムと変換2プログラムをそれぞれ実行することを特徴とする請求

20 項12記載の変換装置。

 16. オフライン一括処理を行う手続き型のプログラムソースコードを入力し、

 このプログラムソースコードをオンライン一件別処理のプログラムに変換し、

- 25 このオンライン一件別処理のプログラムをクライアント／サーバ環境で動作するWebプログラムに変換することを特徴とする変換

方法。

要 約 書

本発明は、既存のプログラムを新しい技術のソフトウェアに適した構造に変換することを目的とする。入力部 1 1 0 0 はコボルプログラム 1 0 0 を入力し、記憶部 1 8 0 0 に記憶する。分割部 1 2 0 0 はコボルプログラム 1 0 0 をまとまりある複数のプログラムに分解し、構文解析部 1 3 0 0 は分解されたそれぞれのプログラムの構文を解析する。プログラム判断部 1 4 0 0 は各プログラムの役割を判断し、節判断部 1 5 0 0 は各プログラム中の節の内容、役割を判断する。これらの動作を終えた後、抽出／変換部 3 1 0 0 は最終プログラム 3 0 0 を生成するためのデータの抽出及び変換を行ない、その結果、コボルプログラム 1 0 0 のソースコードから最終プログラム 3 0 0 のソースコードへの変換が自動的に行われる変換装置を提供する。